Universidade Estadual de Campinas
Faculdade de Engenharia Elétrica e de Computação
Departamento de Comunicações

Grafos-fatores e Decodificação Iterativa: Novas Aplicações

Autor: Alexandre de Andrade
Orientador: Prof. Dr. Jaime Portugheis

Tese de Doutorado apresentada à Faculdade de Engenharia Elétrica e de Computação como parte dos requisitos para obtenção do título de Doutor em Engenharia Elétrica.

Área de Concentração: Telecomunicações e Telemática.

Campinas-SP, Março de 2010.

FICHA CATALOGRÁFICA ELABORADA PELA
BIBLIOTECA DA ÁREA DE ENGENHARIA E ARQUITETURA - BAE - UNICAMP

An24g

Andrade, Alexandre de
Grafos-fatores e decodificação iterativa: novas aplicações / Alexandre de Andrade. --Campinas, SP: [s.n.], 2010.

Orientador: Jaime Portugheis.
Tese de Doutorado - Universidade Estadual de Campinas, Faculdade de Engenharia Elétrica e de Computação.

1. Comunicações digitais. 2. Codificação. 3. Códigos de controles de erros (Teoria da informação). I. Portugheis, Jaime. II. Universidade Estadual de Campinas. Faculdade de Engenharia Elétrica e de Computação. III. Título.

Título em Inglês: Factor graphs and iterative decoding: new developments
Palavras-chave em Inglês: Digital communication, Codification, Error control code (Theory of information)
Área de concentração: Telecomunicações e Telemática
Titulação: Doutor em Engenharia Elétrica
Banca examinadora: Daniel Carvalho da Cunha, Richard Demo Souza, Sueli Costa, Celso de Almeida
Data da defesa: 25/03/2010
Programa de Pós Graduação: Engenharia Elétrica

COMISSÃO JULGADORA - TESE DE DOUTORADO

Candidato: Alexandre de Andrade

Data da Defesa: 25 de março de 2010

Título da Tese: "Grafos-Fatores e Decodificação Iterativa: Novas Aplicações"

Prof. Dr. Jaime Portugheis (Presidente): _____

Prof. Dr. Daniel Carvalho da Cunha: _____

Prof. Dr. Richard Demo Souza: _____

Profa. Dra. Sueli Irene Rodrigues Costa: _____

Prof. Dr. Celso de Almeida: _____

Resumo

Esta tese aborda métodos de estimação probabilística em sistemas de comunicações usando a teoria de grafos-fatores e seu algoritmo genérico soma-produto. Estas ferramentas são atualmente reconhecidas como um ambiente ideal para derivar vários esquemas de decodificação/estimação, e também integrar modelos de componentes típicos num sistema de comunicação para melhor desempenho do processo de recepção. Mais genericamente, são adequadas para projetos de receptores unificados. Além disso, vários esquemas de decodificação conjunta fonte/canal podem ser contextualizados neste ambiente.

Partimos de uma revisão geral da teoria grafos-fatores e do algoritmo soma-produto de forma abrangente e inserindo os fundamentos matemáticos relacionados. Na sequência aplicamos estes conceitos a sistemas de comunicações. Focamos em sistemas de grafos com ciclos, que são de grande interesse e que resultam na versão iterativa do algoritmo soma-produto, onde cronogramas de execução devem ser arbitrados eficientemente.

Descrevemos a decodificação turbo seguindo esquemas de grafos-fatores normais e causais, a forma mais apropriada para a descrição e análise de cronogramas. A partir desta formulação, estudamos o caso da decodificação turbo na sua variante não-bloco, com entrelaçadores periódicos e decodificação contínua/causal. Apresentamos um cronograma completo do algoritmo soma-produto para este caso, mostrando vantagens práticas em relação ao proposto na literatura, sobretudo em relação a sua implementação.

Na última parte da tese, apresentamos um estudo da aplicação de grafos-fatores no problema de decodificação iterativa conjunta fonte/canal. Partimos de um modelo genérico de fonte com memória, discreta no tempo e contínua em amplitude, consideramos quantização vetorial e tratamos o problema da decodificação iterativa conjunta integrando modelos destes componentes com o resto do sistema. Alguns resultados de simulações computacionais para os esquemas propostos são apresentados.

Abstract

The present thesis deals with probabilistic inference methods for communication systems described by the unifying framework of factor graphs and the general elimination algorithm, the so called sum-product algorithm. These exceedingly general tools are understood as a state of the art environment to build many decoding schemes, and to model typical components for a joint efficient inference at reception. More generally, is a suitable framework to unified receiver designs. Additionally, some joint source channel decoding schemes can also be proper modeled under this context.

We start with a review of this framework and related mathematical topics. Thereafter, we particularize to cases of interest, like typical communication systems. This framework gives powerful insights into the structures of multivariate constrained systems and shows how distributed probabilistic inferences can be performed, as shown for typical communication systems with a standard channel encoder. Systems represented by fator graphs with cycles are the most relevant. For the iterative version of the sum–product algorithm, a calculation schedule has to be efficiently chosen.

We review the turbo decoding scheme for the classical turbo code using a normal and causal factor graph realization, providing an environment for scheduling descriptions. Then, we approach the non-block turbo decoding version (stream-oriented turbo codes), where general periodic and causal interleavers can be used and continuous decoding schemes are required. We present a full decoding sum-product schedule for this case, with pratical improvements over the usual non-graphical decoding scheme.

In the last part of this thesis, we address the joint source channel decoding problem using the factor-graph framework. Starting from a general source model, linear discrete time series, we consider straight vectorial quantization. Instead of trying to remove redundancy, we go in the direction of building decoding schemes that explore this redundancy from source model and quantizer map. We analyse cases when iterative decoding can be performed taking these elements into account in a proper way. Some simulation on the proposed schemes are presented.

Agradecimentos

Agradeço ao meu orientador, Jaime Portugheis, pela colaboração fundamental.
Agradeço aos membros da banca, pelas valiosas correções e sugestões.

E agradeço a minha família, por tudo o mais.

Este trabalho foi realizado com o apoio financeiro da Fundação de Amparo à Pesquisa do Estado de São Paulo - FAPESP.

Dedico esta obra aos meus pais

Sumário

Lista de Figuras

Capítulo 1

Introdução

Esta tese tem como tema principal a decodificação probabilística em sistemas de comunicação. E a mais apropriada ferramenta para este estudo são grafos-fatores [54].

O tema conquistou grande interesse com o advento dos códigos turbo [9], e a redescoberta dos códigos LDPC de Gallager [31], com seus respectivos algoritmos de decodificação iterativa. Posteriormente, estes foram descritos e unificados como algoritmos de propagação de mensagens em códigos sobre grafos [64], onde a natureza iterativa provém dos ciclos presentes no grafo.

Gallager [31] foi o primeiro a indicar que códigos de verificação de paridade admitiam naturalmente por construção uma decodificação probabilística iterativa, com operacionalização de baixa complexidade que se estende para qualquer comprimento de bloco.

De acordo com a abstração de Shannon e seu teorema de separação [75], idealmente a tarefa de projetar um sistema de comunicação pode ser separada em duas tarefas independentes: codificação de fonte e codificação de canal.

A princípio, a decodificação iterativa se insere precisamente na teoria de codificação de canal. E de forma providencial, pois fornece um método de baixa complexidade para decodificação de classes de códigos longos e com algum grau de aleatoriedade na sua construção, justamente como Shannon previu serem as características dos melhores códigos.

Podemos adotar a nomenclatura usual códigos da classe turbo (*turbolike*) referindo-se a todos estes esquemas de codificação de canal construídos para admitirem decodificação probabilística iterativa. São códigos cuja representação gráfica possui vários elementos dispersos: vínculos, conexões e ciclos.

Shannon estabeleceu os fundamentos da teoria de informação, formalizou o problema fundamental de comunicação, e determinou os limites da codificação de canal eficiente. Definiu a capacidade de modelos de canais ergódicos.

Desde então, a construção de esquemas de codificação de canal no limite da eficiência, próximos à capacidade de canal, foi um desafio central.

Ao demonstrar o teorema de codificação de canal, onde estabelecia existência e limitantes, Shannon indicou que códigos longos e aleatórios seriam os melhores para atingir os limites de capacidade de canal. Entre as várias dificuldades para operar um código deste tipo, a intransponível por um longo tempo foi a complexidade da decodificação.

Os códigos da classe turbo e sua decodificação iterativa resolveram este problema. Com eles, assintoticamente pode-se chegar tão próximo quanto se desejar dos limites de capacidade de Shannon para modelos de canais clássicos. Esquemas a centésimos de decibéis já foram apresentados. E com complexidade de decodificação praticável. Assim, códigos da classe turbo são códigos factíveis e quase-ótimos.

Todos estes esquemas quase-ótimos são agora entendidos ser códigos definidos sobre grafos, operando com essencialmente um mesmo algoritmo de decodificação, o soma-produto iterativo.

Atualmente, códigos da classe turbo são considerados a solução definitiva para a questão da aproximação de capacidade, e pensar em códigos da classe turbo equivale a pensar em códigos sobre grafos. É a melhor forma de descrever códigos longos, com numerosos vínculos de baixa complexidade, onde o caráter aleatório é dado pela atribuição das numerosas conexões. E a aleatoriedade nas conexões não adiciona complexidade essencial à decodificação.

Assim, estes esquemas de codificação de canal cumpriram a indicação de Shannon para

as características dos melhores códigos: longos e aleatórios. Mas a grande revolução foi, de fato, seu algoritmo de decodificação.

Métodos iterativos sempre permearam as mais variadas técnicas em ciências exatas. Em alguns casos com alguma fundamentação teórica, e em outros apenas heurísticos. Podemos dizer que o algoritmo turbo num primeiro momento pertenceu à segunda situação, até que foi entendido no contexto mais geral dos algoritmos de propagação de confiança em modelos gráficos. A princípio, em redes bayesianas [64], e posteriormente em grafos-fatores [54].

Motivado por seu expressivo desempenho, numa aproximação empírica dos limites de capacidade sem precedentes, a busca por uma fundamentação teórica da decodificação turbo tem sido intensa. Ao se encaixar no modelo de grafos e seus algoritmos de passagem de mensagens, um grande avanço neste sentido foi conquistado. Seus cálculos são de natureza exata em grafos sem ciclos, e podem ser entendidos como aproximações em grafos com ciclos.

O mais notável a respeito de modelos gráficos é sua naturalidade em formular modelos probabilísticos de fenômenos complexos em diversas áreas de conhecimento onde estes se aplicam, enquanto a complexidade do modelo é mantida sob controle.

Na teoria moderna de códigos de controle de erros, modelos gráficos se tornaram uma ferramenta essencial. A sua utilidade se torna evidente quando o problema da decodificação é considerado. De uma forma geral, a decodificação pode ser pensada como sendo essencialmente um problema de inferência estatística.

Enquanto modelos clássicos de construção de códigos visando decodificação abrupta (*hard decoding*) são modelos algébricos, códigos construídos para decodificação suave (*soft decoding*) são melhor descritos e construídos a partir de modelos gráficos, sobre os quais algoritmos probabilísticos podem operar. Algoritmos com cálculos locais e distribuídos (portanto, de baixa complexidade).

Assim, a teoria de códigos sobre grafos é uma das ramificações da teoria moderna de códigos.

A teoria turbo redirecionou o foco das atenções mais para a decodificação do que a codificação, que pôde ser colocada como um mero instrumento, com poucas estruturas

regulares ou complexas, e mais voltada para a definitiva finalidade da decodificação. Uma decodificação probabilística, que serve para qualquer código.

O algoritmo soma-produto (SP) generalizado engloba todos os algoritmos que operam sobre modelos gráficos estabelecidos anteriormente, unificando-os e evidenciando que todos são relacionados, por serem casos particulares, instâncias de seu método geral. Todos estes esquemas sub-ótimos são agora entendidos ser códigos definidos sobre grafos, operando com essencialmente um mesmo algoritmo de decodificação, o soma-produto iterativo.

Embora os principais conceitos da teoria de grafos-fatores e algoritmo SP já estivessem presentes na teoria de propagação de probabilidades em redes bayesianas, esta teoria trouxe uma nova formulação mais esclarecedora, identificando conceitos e elementos atuantes de tal forma que o algoritmo de passagem de mensagens ganhou uma descrição mais simples e intuitiva.

A descrição dos cálculos das mensagens é mais simples em relação ao algortimo BP (*belief propagation*) de redes bayesianas. E o sistema base representado pelo grafo não é necessariamente probabilístico. Podemos dizer que o algoritmo BP é também uma instância do SP.

Grafos-fatores fornecem uma representação mais objetiva em relação a modelos gráficos anteriores, e possibilitam englobar algoritmos de inferência mais gerais. Assim, a teoria de grafos-fatores unificou de forma elegante modelos gráficos anteriores a ela.

A origem da decodificação iterativa vem da área de codificação de canal. Entretanto, o método turbo pode ser adaptado e aplicado nas diversas partes de um sistema de comunicação, com memória e/ou redundância, intencionais ou circunstanciais, que podem interagir conjuntamente numa decodificação probabilística. Atualmente, a nomenclatura "decodificação iterativa" pode ser atribuída a qualquer aplicação do princípio turbo, nos mais diversos componentes de um sistema. Assim, já consta o princípio turbo aplicado, por exemplo, à equalização e à decodificação conjunta fonte/canal.

Se num primeiro momento este inseriu-se na codificação de canal subótima, com códigos longos e irregulares, depois percebeu-se que mesmo para sistemas fora deste contexto seria

útil a estimação iterativa de algumas variáveis, já que na essência tanto redundância quanto memória, os elementos presentes a qualquer sistema de comunicação, podem ser aproveitados por este tipo de decodificação. Vários estudos seguiram neste sentido, tais como [1] [10] [11] [13] [14] [17] [34] [36] [39] [44] [67] [70] [76] [87] e [88].

Utilizando a modelagem por grafos-fatores, esta integração fica expressivamente mais compreensível e prática. Fica assim estabelecido o conceito de receptor iterativo unificado [88], que consiste na modelagem de vários componentes típicos de sistemas de comunicação num grafo-fator global, representando todos os acoplamentos entre eles. Assim, mais do que uma decodificação, é possível uma inferência generalizada e integrada do sistema, eventualmente iterativa e aproximada quando na presença de ciclos.

Em geral, grafos-fatores podem modelar todos os componentes típicos de um sistema de comunicação, sobretudo quando admitem modelos de vínculos simples para suas variáveis. Portanto, um receptor iterativo unificado baseado em grafos-fatores pode integrar vários problemas típicos que são usualmente tratados separadamente, como estimação de canal, equalização, cancelamento de interferência, decodificação de fonte, etc.

1.1. Sobre o conteúdo da tese

As principais contribuições desta tese são novos entendimentos e novas observações relevantes na área específica de receptores iterativos generalizados, descritos por grafos-fatores.

Como o foco aqui é decodificação e modelos de sistemas sobre grafos, deixamos de lado o material clássico de teoria de codificação de canal, inserindo apenas os conceitos e resultados necessários para o entendimento do tema.

Os pré-requisitos para a leitura deste trabalho são os conceitos básicos de teoria de probabilidades, teoria da informação, teoria de codificação e teoria de comunicação.

Exaustivas simulações com diferentes parâmetros foram feitas durante este projeto de doutorado. A tese apresenta apenas algumas selecionadas por sua relevância e/ou representatividade.

1.2. Principais contribuições da tese

- A modelagem e descrição do cronograma de decodicação no contexto de grafos-fatores para um esquema turbo *stream-oriented*, com vantagens práticas em relação ao esquema tradicional dado na literatura;

- A integração da quantização vetorial numa decodificação iterativa fonte/canal conjunta no contexto de grafos-fatores. Além disso, apresentamos uma revisão de como vem sendo feita a decodificação iterativa fonte/canal conjunta na literatura existente, contextualizando no ambiente de grafos-fatores;

Maiores detalhes sobre estas e outras contribuições contidas em cada capítulo são dados ao final de cada um deles. O desenvolvimento do esquema turbo *stream-oriented* foi apresentado no XXI Simpósio Brasileiro de Telecomunicações. Parte do capitulo 4 foi utilizado na elaboração de minicurso ministrado pelo orientador.

Capítulo 2

Grafos-fatores e o algoritmo soma-produto

Este capítulo é dedicado à introdução e análise das ferramentas centrais de nosso estudo, a teoria de grafos-fatores e o algoritmo genérico associado, o soma-produto (SP). As principais referências nestes tópicos são [53], [84], [85], [88], [29], e principalmente [54]. O objetivo deste capítulo é apresentar uma compilação sobre teoria de grafos-fatores, com alguns desenvolvimentos adicionais não existentes na literatura. O texto não tem a pretensão de exaurir todas as possibilidades do tema, e sim de servir como referência para os capítulos seguintes. Entretanto, incluídas aqui estão algumas pequenas contribuições para um melhor entendimento desta teoria, com interpretações e observações relevantes e que não encontramos em parte alguma da literatura. Podemos destacar algumas delas: uma análise sobre a definição de grafos-fatores e sua abrangência não algébrica; uma demonstração formal para as expressões recursivas do algoritmo SP; uma definição mais precisa para o que a literatura chama de "projeção" entre espaços e sub-espaços de configurações; uma melhor intepretação sobre o papel das variáveis observadas no algoritmo de marginalização; e um foco maior na interpretação do SP por medidas e funções custos, que entendemos fornecer a melhor heurística para a teoria; e uma interpretação dos elementos do algoritmo clássico *forward-backward* (ou BCJR) utilizando a teoria de grafos-fatores.

2.1. Introdução

De uma forma geral, problemas relevantes de estimações (decisões sob ambientes de incerteza) em diversas áreas, tais como processamento de sinais, física estatística, inteligência artificial, além da teoria da códigos corretores de erros para comunicação através de canal ruidoso, podem ser reformuladas matematicamente como um problema de marginalização e/ou otimização de uma função global do sistema, esta sendo obtida a partir da composição de funções locais associadas a fragmentos do sistema. A função global é uma função custo que associa um valor a cada configuração do sistema, e dependendo do caso pode ser uma medida de potencial, de energia, de probabilidade, confiança, etc, e o problema dado consiste em obter um ponto de otimalidade - um mínimo ou um máximo, dependendo de sua interpretação - desta medida global, ou das medidas marginais derivadas a partir dela.

A teoria de grafos-fatores e seu algoritmo genérico de marginalização associado, o soma-produto (SP), traz uma forma elegante e intuitiva de tratar problemas deste tipo [54].

Grafos-fatores representam composições de funções multivariadas que compartilham a mesma coleção de variáveis. O problema geral de marginalização de composições deste tipo pode ser visualizado sumariamente através destes grafos, já que todos os principais elementos, funções e variáveis, estão presentes e devidamente rotulados nestes grafos. Mais do que isso, eles representam graficamente e de forma sumária o esquema de um algoritmo genérico de marginalização, o algoritmo SP.

A abordagem por grafos-fatores engloba e generaliza outros modelos gráficos anteriores a ele: "redes bayesianas" (*Bayesian Networks*) e "campos aleatórios markovianos" (*Markov Random Fields*) , que operam em redes de variáveis aleatórias. Embora não exatamente equivalentes, estes modelos surgem da idéia de representar graficamente as dependências e independências locais de variáveis aleatórias num sistema. E ambos resultam em fatorações da função de distribuição conjunta de probabilidade, sendo expressa como o produto de distribuições de probabilidade envolvendo subconjuntos de variáveis, com probabilidades condicionais em algumas delas.

A modelagem por grafos-fatores generaliza esta idéia e não se limita a representar

distribuições de probabilidades. Variáveis, funções fatores, e suas associações são os elementos requeridos na definição de um grafo-fator. Logo, é possível construir um sistema a partir destes elementos, dos quais a função global resultante é consequência.

Um sistema modelado em várias variáveis e funções medidas deste tipo inclui casos onde as funções dadas representam vínculos genéricos entre subconjuntos de variáveis. A qualquer vínculo, determinístico ou probabilístico, pode ser associada uma função custo.

Uma vez estabelecidas medidas, é natural que inferências sobre uma parte das variáveis sejam obtidas através da eliminação por marginalização do restante de variáveis do sistema.

No caso particular da teoria de códigos, o modelo clássico de representação gráfica de códigos são as treliças. Todos os modelos gráficos, redes bayesianas, campos markovianos, treliças para códigos, tem algoritmos associados, seja de propagação de probabilidades, ou mais genericamente, algoritmos distributivos operando baseados no modelo gráfico. A teoria de grafos-fatores e SP unifica e generaliza todos estes modelos e respectivos algoritmos, esclarecendo que vários deles em última instância consistem em algum tipo de marginalização.

No caso particular de problemas de decodificação símbolo-a-símbolo em sistemas de comunicação, que são a base para os vários tipos de decodificação iterativa, o problema equivale a obter distribuições marginais simples (em variáveis únicas) de uma distribuição conjunta. Ou seja, decodificar uma dada variável significa obter sua distribuição marginal de probabilidade, e realizar a decisão (estimativa) de seu valor a partir desta marginal.

Problemas de inferência probabilística envolvem duas classes de elementos: vínculos e observações *a posteriori* de variáveis. De forma geral, temos um sistema modelado em várias variáveis aleatórias e vínculos estabelecidos entre elas, e desejamos inferir os valores de algumas destas variáveis a partir da observação de outras. A inferência ótima de cada variável é sempre relacionada à obtenção de sua distribuição marginal a partir de uma distribuição conjunta global do sistema, desde que esta distribuição conjunta considere todos os vínculos e observações *a posteriori* de variáveis. As duas classes, vínculos e observações, são então modeladas como funções que vão compor a função global do sistema. Vínculos probabilísticos são associados a distribuições de probabilidades condicionais. Vínculos determinísticos e

observações de variáveis são associados a funções indicadoras.

Diferentemente da descrição de códigos por treliças, que se preocupa em representar graficamente o código completamente, a representação por grafos-fatores é mais compacta e sumária, representando apenas uma estrutura essencial sobre símbolos, estados, vínculos e conexões entre eles para uma dada realização de um código através deles, já que esta estrutura indica de forma natural e intuitiva como se procede a codificação e decodificação.

Uma vez que um dado sistema é modelado por funções-fatores e uma função global é estabelecida, o algoritmo SP é um algoritmo genérico que opera sobre o grafo-fator associado para se extrair as marginais. Embora o algoritmo SP possa ser descrito independentemente da abordagem por grafos-fatores, ele fica significativamente mais intuitivo e elegante desta forma.

Os casos mais notórios de algoritmos que são instâncias do algoritmo genérico SP são: o algoritmo de Viterbi [81], o algoritmo *forward-backward* (ou BCJR [4]), o algoritmo BP (*belief propagation*) de inteligência artificial [52] [68], e a decodificação turbo [9], que equivale ao SP aplicado num grafo com ciclos.

2.2. Marginalização

De maneira geral, para uma função de várias variáveis $g(x_1, \cdots, x_n)$, a operação de marginalização em uma variável x_i é dada por

$$\underset{x_i}{\operatorname{marg}} g(x_1, \cdots, x_n) = \sum_{x_1} \cdots \sum_{x_{i-1}} \sum_{x_{i+1}} \cdots \sum_{x_n} g(x_1, \cdots, x_n), \tag{2.1}$$

ou ainda, de acordo com a notação de [54],

$$\sum_{\sim\{x_i\}} g(x_1, \cdots, x_n) = \sum_{x_1} \cdots \sum_{x_{i-1}} \sum_{x_{i+1}} \cdots \sum_{x_n} g(x_1, \cdots, x_n), \tag{2.2}$$

com o "operador sumário" na variável x_i, que denota a operação de soma em todas as variáveis exceto x_i. Por definição, o cálculo (2.1) acima deve ser entendido como sendo realizado ponto a ponto. O resultado é uma função simples (em uma variável), na variável x_i.

Visto como um operador atuando sobre a função g, o operador sumário (ou operador de marginalização) é uma composição de operadores mais simples. Podemos ainda abstrair o conceito de 'somatório em uma dada variável em todo seu domínio' como um 'operador'. Ele opera sobre funções de várias variáveis, e tem o efeito de eliminar uma delas. Dependendo do tipo de domínio, um conjunto discreto ou contínuo, o operador significa, respectivamente, uma soma finita ou uma integração. Além disso, este operador assume diferentes significados quando consideramos marginalizações em diferentes semi-anéis algébricos. Outro fato relevante é que operadores deste tipo comutam entre si. Portanto, o operador sumário é uma composição comutativa destes operadores.

O conceito de grafo-fator é associado a funções de várias variáveis e suas eventuais representações como produtos de funções mais simples. Por exemplo, dada uma função $g(x_1, x_2, x_3, x_4, x_5)$, se existem f_A, f_B, f_C, f_D, f_E tais que

$$g(x_1, x_2, x_3, x_4, x_5) = f_A(x_1) f_B(x_2) f_C(x_1, x_2, x_3) f_D(x_3, x_4) f_E(x_3, x_5) \tag{2.3}$$

então dizemos que g se fatora nas funções f_A, f_B, f_C, f_D, f_E, e que estas são funções-fatores da função global g.

Expressões deste tipo denotam igualdade entre funções, indexando variáveis correspondentes nos dois lados da igualdade para significar que cada variável tem domínio bem definido e a igualdade vale no domínio global envolvendo todas as variáveis.

A marginalização da função $g(x_1, x_2, x_3, x_4, x_5)$ em uma dada variável, digamos x_3, consiste no seguinte cálculo

$$\begin{aligned} &\textstyle\sum_{x_1}\sum_{x_2}\sum_{x_4}\sum_{x_5} g(x_1, x_2, x_3, x_4, x_5) = \\ &\qquad \textstyle\sum_{x_1}\sum_{x_2}\sum_{x_4}\sum_{x_5} f_A(x_1) f_B(x_2) f_C(x_1, x_2, x_3) f_D(x_3, x_4) f_E(x_3, x_5) \end{aligned} \tag{2.4}$$

e pode ser simplificada apenas utilizando as propriedades comuns a semi-anéis algébricos, tais como comutatividade e distributividade. A única característica necessária é que as funções-fatores dependam de menos variáveis do que a função global. A abordagem por grafos-fatores cria um método sistemático para identificar estas simplificações.

2.3. Grafos: conceitos básicos

É conveniente aqui apresentar algumas definições básicas da teoria de grafos, pelo menos as que são úteis ao que se segue.

Definição Um grafo $G = (V, E)$ consiste em um conjunto finito não vazio V, o conjunto de **nós** (*node*), e um conjunto E de **ramos** (*edges*) entre pares de elementos de V.

Ramos também podem ser chamados de **conexões**, ou ainda **arestas**, e nós também podem ser denomidados **vértices**.

Podemos identificar os ramos de um grafo como sendo subconjuntos de dois elementos de V, do tipo $\{v_1, v_2\}$. Portanto, na teoria abstrata de conjuntos um grafo pode ser caracterizado por um conjunto V e um conjunto de subconjuntos de dois elementos de V.

Por simplificação podemos denotar $uv \in G$ para a existência de um ramo $\{u, v\}$ no grafo G, e dizemos que u e v são **vizinhos** (*neighbours*). Denotamos $N(u)$ o conjunto de vizinhos de um elemento, isto é, $N(u) = \{v \in G : uv \in G\}$.

Dizemos que há um **caminho** entre dois nós u_0 e u_n se existem conexões $u_0u_1, \ldots u_{n-1}u_n \in G$. Neste caso, n é o **comprimento** do caminho. Um grafo é dito **conexo** se há pelo menos um caminho entre cada par de nós. A **distância** entre dois nós $d(u, v)$ é o valor do menor caminho entre u e v. Para um grafo conexo, podemos definir seu **diâmetro** como sendo a maior distância entre nós existentes.

Um **ciclo** num grafo é um caminho de um elemento u para o próprio u. Dizemos que um grafo tem ciclos se existe pelo menos um nó com um ciclo. Um grafo é uma **árvore** se é conexo e não tem ciclos.

Um **grafo bipartido** em X e Y é um grafo tal que $X \cap Y = \emptyset$, $X \cup Y = V$, e para todo $uv \in G$, $u \in X \Leftrightarrow v \in Y$. Isso significa dizer que há dois tipos de nós, tipo X e tipo Y, e os vizinhos de cada nó são de tipo diferente dele.

2.4. Grafos-fatores: definição

O conceito de um grafo-fator é auto-explicativo e intuitivo, podendo ser ilustrado por um simples exemplo. Considerando o caso de uma função $g(x_1, x_2, x_3, x_4, x_5)$ que admite a seguinte fatoração

$$g(x_1, x_2, x_3, x_4, x_5) = f_A(x_1) f_B(x_2) f_C(x_1, x_2, x_3) f_D(x_3, x_4) f_E(x_3, x_5), \qquad (2.5)$$

o grafo-fator associado a ela é esquematizado na figura 2.1 a seguir.

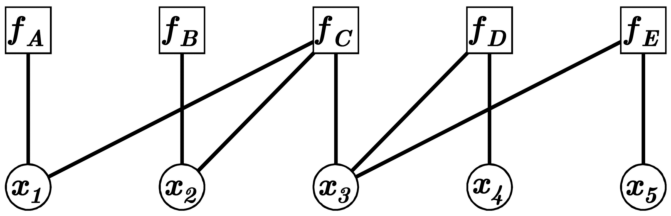

Figura 2.1: Exemplo de grafo-fator.

Um grafo-fator descreve uma coleção de funções que compartilham a mesma coleção de 'argumentos'. É um grafo bipartido em **nós-variáveis** e **nós-funções**. A convenção para caracterizar seus elementos gráficos é: nós-variáveis são representadas por círculos; e nós-funções por retângulos. Um nó-variável e um nó-função são conectados por um ramo (não direcionado) se e só se a variável é argumento da função. Como há correspondências biunívocas entre nós do grafo e os elementos matemáticos funções e variáveis, é comum não fazer distinção entre um nó e a entidade que ele representa para simplificar a linguagem.

Num caso genérico, cada uma das funções-fatores tem como argumentos um subconjunto das variáveis x_i. Uma expressão genérica para este tipo de fatoração requer uma notação que enumere as variáveis de cada função e seus respectivos conjuntos domínios. O ponto de partida para uma definição geral é a definição do domínio da função global, o chamado **espaço de configurações**.

O espaço de configurações associado à sequência de conjuntos $A_1, A_2, \ldots, A_n$ é o produto

cartesiano

$$A_1 \times A_2 \times \ldots \times A_n, \tag{2.6}$$

também denotado

$$\bigotimes_{i=1}^{n} A_i. \tag{2.7}$$

O espaço de configurações é o domínio da função global da ocasião.

Para um subconjunto de índices $I = \{i_1, i_2, \ldots, i_r\} \subseteq \{1, \ldots, n\}$, o produto cartesiano $A_{i_1} \times A_{i_2} \times \ldots \times A_{i_r}$ é dito um **sub-espaço de configurações**. Portanto, para cada subconjunto de índices $I \subseteq \{1, \ldots, n\}$, temos um sub-espaço associado, e denotamos $A_I = \bigotimes_{i \in I} A_i$. Cada função-fator tem como domínio um sub-espaço de configurações.

Podemos ainda estender o conjunto de índices $\{1, \ldots, n\}$ para todo o conjunto dos números naturais $\mathbb{N}$, que é equivalente a considerar conjuntos discretos (contáveis) de índices. Já para os sub-espaços, é suficiente considerar apenas produtos cartesianos finitos (I finito).

Aqui também identifica-se $A \times B \equiv B \times A$, e apenas produtos cartesianos ordenados no conjunto de índices são considerados. As variáveis e seus domínios são enumerados e ordenados respectivamente.

Quando restrito a códigos, os conjuntos A_i são alfabetos, espaços de estados de cada símbolo. Neste caso, são conjuntos finitos. No caso geral, A_i pode ser qualquer, como por exemplo o conjunto de números reais $\mathbb{R}$.

No que se segue, consideramos um conjunto fixo R como contradomínio para as funções.

Dado um espaço de configurações, uma **função-nó** é uma função definida em algum sub-espaço, ou seja, é uma função $f : \bigotimes_{i \in I} A_i \to R$ para algum subconjunto finito de índices I.

Uma coleção finita de funções nós sobre um espaço de configurações é tudo que precisamos para caracterizar uma fatoração. Entretanto, para definir um grafo-fator associado, há o conceito auxiliar de **variável**.

Para cada A_i, seja x_i uma variável assumindo valores em A_i. Denotamos um elemento de $A_1 \times A_2 \times \ldots \times A_n$ por $(x_1, x_2, \ldots, x_n)$, ou simplesmente $\mathbf{x}$. Similarmente, $\mathbf{x}_I =$

$(x_{i_1}, x_{i_2}, \ldots, x_{i_r})$ é um elemento de um dado subespaço A_I. Desta forma, para uma função-fator f definida no sub-espaço $A_{i_1} \times A_{i_2} \times \ldots \times A_{i_r}$, denotamos $f(x_{i_1}, x_{i_2}, \ldots, x_{i_r})$ ou $f(\mathbf{x}_I)$ para representar a dependência em relação às suas variáveis. Dizemos ainda que f tem como argumentos as variáveis $x_{i_1}, x_{i_2}, \ldots, x_{i_r}$, ou que depende destas variáveis.

O conceito de variável, que pode ser entendido como um indexador para explicitar argumentos de funções, aqui vai além disso e resulta na entidade que compõe o grafo-fator.

Definição Para um conjunto de variáveis $\mathbf{X} = (x_1, x_2, \ldots, x_n)$ e um conjunto de funções $\mathbf{F} = (f_1, f_2, \ldots, f_m)$ com argumentos em $\mathbf{X}$, cada f_j com argumentos $\mathbf{x}_{I_j}$, um grafo-fator é um grafo G bipartido com vértices em $\mathbf{X}$ e $\mathbf{F}$ ($\mathbf{X}$,$\mathbf{F}$-bipartido) que representa a fatoração

$$g(\mathbf{x}) = \prod_{j=1}^{m} f_j\left(\mathbf{x}_{I_j}\right), \tag{2.8}$$

com $x_i f_j \in G$ se e somente se x_i é argumento de f_j.

Desta forma, no grafo, o conjunto $N(f_j)$ de vizinhos de f_j são as variáveis argumentos de f_j. E $N(x_i)$ são as funções que tem x_i como argumento.

Portanto, por definição um grafo-fator é um grafo bipartido que representa uma fatoração de uma dada função, onde há um nó para cada variável e um nó para cada função, e um ramo para cada dependência entre função e variável. Para duas coleções, uma de funções e uma de variáveis, as conexões do grafo-fator definem a relação "é argumento de" entre seus elementos.

Nesta definição usual de grafos-fatores, é presumido que o contradomínio R tem uma estrutura algébrica, um semi-anel comutativo. Entretanto, a noção de grafo-fator tem como finalidade apenas descrever as relações entre variáveis e funções, e o produto dado na definição não seria sequer necessário. Numa definição mais abrangente, trata-se de um grafo que nos vértices enumera as sequências de variáveis $x_1, x_2, \ldots, x_n$ e de funções $f_1, f_2, \ldots, f_m$, e conecta x_i a f_j se e somente se x_i é argumento de f_j. O fato de que as funções $f_1, f_2, \ldots, f_m$ são combinadas algebricamente através da operação multiplicativa de um dado semi-anel poderia ser estabelecido em separado.

O fato de que a combinação de funções custos se dá através de uma operação

que é distributiva em relação à operação marginalizadora é essencial. Um problema de marginalização tem sempre uma operação algébrica de "adição" associada. E a combinação de funções por uma operação algébrica "produto" (que seja distributiva em relação à "adição") é necessária para a aplicação do algoritmo SP.

2.5. Conceitos de álgebra abstrata

Nos fundamentos da álgebra formal, as estruturas algébricas definidas mais comuns, ordenadas da mais simples para a mais completa, são: semigrupos, monóides, grupos, semi-anéis, anéis, e campos (ou corpos). Nesta sequência de classes de estruturas, uma dada classe é mais restritiva e forma um subconjunto da anterior, ou seja, todo campo é um anel, todo anel é um semi-anel, e assim por diante.

As estruturas mais simples - semigrupos, monóides e grupos - tem sutis diferenças entre elas. Monóides são mais simples que grupos por não exigir a existência de elementos inversos. Semigrupos são mais simples que monóides por não exigir a existência de um elemento neutro. Todas estas estruturas tem como característica mínima a associatividade. E por fim, em todas estas estruturas algébricas, a comutatividade é um adendo opcional e desejável, mas não necessário.

Definimos formalmente a seguir as estruturas algébricas relevantes no nosso contexto.

Um **monóide comutativo** consiste em

- um conjunto não vazio M;
- uma 'operação binária associativa' nele definida, que formalmente é uma função $S : M \times M \to M$ tal que $S\left(S\left(m_1, m_2\right), m_3\right) = S\left(m_1, S\left(m_2, m_3\right)\right)$ para quaisquer elementos $m_1, m_2, m_3 \in M$;
- a existência de um elemento $m \in M$ (elemento neutro ou elemento identidade) tal que $S\left(m, \cdot\right)$ e $S\left(\cdot, m\right)$ resultam na função identidade em M;

- e na equivalência $S(m_1, m_2) = S(m_2, m_1)$ para quaisquer $m_1, m_2 \in M$.

A propriedade associativa é equivalente a desejar que a operação básica S possa ser estendida naturalmente para qualquer número finito de argumentos, e que estará bem definida, sem ambiguidades. Por exemplo,

$$S(m_1, m_2, m_3, m_4) \triangleq S(S(S(m_1, m_2), m_3), m_4). \tag{2.9}$$

Sendo M^n o n-ésimo produto cartesiano de M, $S : M^n \to M$ está bem definida, a partir de $S : M^2 \to M$ e a associatividade. O cálculo consiste em agrupar termos aos pares, e compor $n-1$ vezes a função S. A associatividade garante unicidade no cálculo, independentemente da escolha da ordem dos agrupamentos de pares.

A partir da definição de monóides, podemos então definir uma estrutura algébrica mais completa: um semi-anel comutativo, que é um conjunto com duas estruturas de monóide comutativo, onde as duas operações se relacionam pela lei distributiva.

Precisamente, um **semi-anel comutativo** consiste

- em um conjunto não vazio R;
- em duas operações monoidais comutativas S e P sobre R;
- na inter-relação entre S e P dada pela equivalência

  $$P(r, S(r_1, r_2)) = S(P(r, r_1), P(r, r_2))$$

  para quaisquer $r, r_1, r_2 \in R$.

Na forma mais usual, pela notação mesofixa, as operações binárias são denotadas por símbolos operadores, tal como

$$r_1 \oplus r_2 \triangleq S(r_1, r_2) \tag{2.10}$$

e

$$r_1 \otimes r_2 \triangleq P(r_1, r_2). \tag{2.11}$$

Assim, a definição usual é a seguinte:

Um **semi-anel comutativo** é uma estrutura algébrica $(R, \oplus, \otimes)$, com um conjunto R e duas operações binárias $\oplus$ e $\otimes$ associativas e comutativas, cada uma com um elemento neutro, $\mathbf{0}$ para $\oplus$ e $\mathbf{1}$ para $\otimes$, tal que $\otimes$ é distributiva sobre $\oplus$, isto é, para quaisquer elementos $r, r_1, r_2 \in R$ vale

$$r \otimes (r_1 \oplus r_2) = (r \otimes r_1) \oplus (r \otimes r_2) \tag{2.12}$$

Evidentemente, qualquer campo ou anel é caso particular de um semi-anel. A diferença de um anel para um semi-anel é a existência de inversos aditivos. Portanto, são exemplos de semi-anéis, considerando suas operações de adição e multiplicação usuais: o campo dos números reais $(\mathbb{R}, +, \cdot)$, o anel dos inteiros $(\mathbb{Z}, +, \cdot)$ e toda a classe de anéis de inteiros módulo n, usualmente denotados $\mathbb{Z}_n$, com $n \in \mathbb{N}$. Os inteiros não negativos $(\mathbb{Z}^+, +, \cdot)$ e os reais não negativos $(\mathbb{R}^+, +, \cdot)$ são apenas semi-anéis e não são anéis.

Um exemplo menos trivial de semi-anel com respeito às operações é $(\mathbb{R}^+, \text{máx}, \cdot)$. De fato, a partir das operações máx e mín combinadas com a soma e produto usuais, é possível construir vários semi-anéis (distintos, mas alguns deles isomorfos entre si) escolhendo o conjunto adequado, já que há distributividade entre estas operações em alguns subconjuntos de $\mathbb{R}$ e $\mathbb{Z}$. Por exemplo,

$$\text{máx}\,(a \cdot b, a \cdot c) = a \cdot \text{máx}\,(b, c) \text{ em } \mathbb{R}^+ \text{ ou } \mathbb{Z}^+ \tag{2.13}$$

$$\text{mín}\,(a + b, a + c) = a + \text{mín}\,(b, c) \text{ em } \mathbb{R} \text{ ou } \mathbb{Z} \tag{2.14}$$

$$\text{máx}\,(\text{mín}\,(a, b), \text{mín}\,(a, c)) = \text{mín}\,(a, \text{máx}\,(b, c)) \text{ em } \mathbb{R} \text{ ou } \mathbb{Z} \tag{2.15}$$

Satisfeita a distributividade, eventualmente para desempenhar o papel de elemento neutro e completar o semi-anel para estas operações, é necessário incluir os elementos $+\infty$ ou $-\infty$. Temos, por exemplo, o semi-anel $(\mathbb{R} \cup \{-\infty, +\infty\}, \text{máx}, \text{mín})$, com $-\infty$ neutro para a "adição" máx do anel, e $+\infty$ para "multiplicação" mín.

Para qualquer conjunto A, se R é um semi-anel, considerando o conjunto todas as funções $f : A \to R$, temos um semi-anel de funções. É usual denotar o conjunto destas funções como R^A. É neste tipo de semi-anel que o algoritmo SP opera.

Para um campo $(F, \oplus, \otimes)$, vale

$$x \otimes y = \mathbf{0} \text{ se e só se } x = \mathbf{0} \text{ ou } y = \mathbf{0}. \tag{2.16}$$

No caso de anéis e semi-anéis, essa regra de cancelamento nem sempre vale. Entretanto, em todos os exemplos de semi-anéis nos casos usuais de aplicação do algoritmo SP, esta propriedade está presente. Esta é uma propriedade requerida para caracterizar 'medidas', ou 'funções custos'.

Para o SP iterativo, uma forma válida de se iniciar o algoritmo é considerar que todas as mensagens iniciais são iguais ao elemento neutro da adição no semi-anel de funções, a função constante $f(x) = 1, \forall x$. Esta é a 'função custo neutra'.

2.6. A lei distributiva

A marginalização de composições representadas por grafos-fatores envolve somas de produtos de funções, e a lei de comutação entre os operadores soma e produto da álgebra, mais conhecida como lei distributiva, desempenha um papel fundamental no algoritmo SP.

Formalmente, a lei distributiva estabelece a regra que rege a comutação entre as duas operações algébricas, soma e produto. Quando generalizada para somatórios e produtórios, a lei distributiva pode ser formulada: para subconjuntos finitos $S_1, \ldots, S_k$ em um semi-anel,

$$\prod_{j=1}^{k} \sum_{v_j \in S_j} v_j = \sum_{(v_1,\ldots,v_k) \in S_1 \times \ldots \times S_k} \prod_{j=1}^{k} v_j \ . \tag{2.17}$$

Outras distributividades podem ser expressas de forma análoga. Por exemplo, a distributividade da operação 'produto' sobre a operação 'máximo' em $\mathbb{R}^+$ pode ser formulada como

$$\prod_{j=1}^{k} \max_{v_j \in S_j} v_j = \max_{(v_1,\ldots,v_k) \in S_1 \times \ldots \times S_k} \prod_{j=1}^{k} v_j \ . \tag{2.18}$$

Associado ao semi-anel com as operações máximo e produto é o problema geral de inferência da "explicação mais provável", associado à maximização da função global, que resulta na variação máximo-produto do algoritmo distribuitivo, do qual o algoritmo de Viterbi operando sobre treliças é um caso particular. Neste caso, o algoritmo SP pode operar de forma diferente da usual, e ao eliminar uma variável por marginalização, calcular e propagar um dos dois elementos (ou os dois se for de interesse): o valor máximo associado ou o ponto de máximo associado à variável. O objetivo é a obtenção do valor máximo e/ou o ponto do espaço de configurações que maximiza a função global.

O algoritmo SP utiliza a distributividade como equivalência para transformar marginalizações globais em marginalizações locais. Ou seja, ele opera no sentido inverso ao da lei distributiva transformando, por exemplo, $xz + yz$ em $(x + y)\, z$.

Já considerando esta ordem inversa para expressar as equivalências, para funções $f : A_1 \to R$ e $g : A_2 \to R$ podemos formular a lei distributiva como

$$\sum_{(x,y)\in A_1\times A_2} f(x)\, g(y) = \sum_{x\in A_1} f(x) \sum_{y\in A_2} g(y) \tag{2.19}$$

ou ainda, ilustrando a invariância em relação a outras variáveis, sendo $f : A_1 \times A_3 \to R$ e $g : A_1 \times A_4 \to R$ arbitrárias,

$$\sum_{(x,y)\in A_1\times A_2} f(x,w)\, g(y,z) = \sum_{x\in A_1} f(x,w) \sum_{y\in A_2} g(y,z)\,. \tag{2.20}$$

É este tipo de transformação que o algoritmo SP realiza, apenas sistematizando todas as possíveis simplificações deste tipo que uma função global admite.

2.7. O algoritmo soma-produto

A partir de uma função $g(x_1, \ldots, x_n)$ de n variáveis, podem ser derivadas até $2^n - 1$ funções marginais distintas, considerando as marginais simples (em uma variável) e as marginais compostas (em mais de uma variável). Um problema de inferência num sistema que tem esta função global pode envolver o cálculo ponto a ponto de uma ou mais destas

marginais. O algoritmo SP estabelece um procedimento eficiente para este tipo de cálculo quando há uma fatoração disponível. Podemos nos restringir à descrição dele para a obtenção de marginais simples, uma vez que a extensão para marginais compostas é natural.

O algoritmo SP estabelece que qualquer marginalização sobre um grafo-fator pode ser realizada pelo cálculo recursivo de um conjunto de funções simples (dependentes de variáveis únicas), que podem ser entendidas como mensagens que se propagam ao longo do grafo.

Assim como as funções-fatores podem ser entendidas como funções custos associadas aos nós no grafo, o algoritmo SP é o processo sistematizado de obter funções custos simples associadas a cada ramo do grafo-fator. Desta forma, cada nó recebe como mensagem de um dado ramo uma função custo simples, para dar continuidade aos cálculos seguintes. Quando considerado um ramo e uma direção, é possível definir o fluxo local de mensagens relativo a eles, ou seja, as mensagens "anteriores".

Pelo caráter bipartido do grafo, a cada ramo do grafo apenas uma variável está associada (a que participa do par deste ramo). As mensagens que passam por este ramo são funções desta única variável. Além disso, se consideramos o fluxo sequencial, a função custo transmitida por um ramo no grafo pode ser descrita simplesmente como: "o produto das funções custos dos ramos 'anteriores' do fluxo, composto com a função custo do nó, marginalizado para se obter uma função custo resultante na variável".

Para um dado ramo (x, f), são associadas duas mensagens, uma em cada direção do ramo. Denotando $\mu_{x\to f}$ a mensagem enviada pela variável x para a função f, e $\mu_{f\to x}$ a mensagem na direção inversa, estas duas mensagens são função de uma variável, de x. As figuras 2.2 e 2.3 a seguir esquematizam os elementos envolvidos no cálculo destas duas mensagens. As funções vizinhas de x são enumeradas h_1, h_2, ... além de f. As variáveis vizinhas de f são enumeradas w_1, w_2, ... além de x.

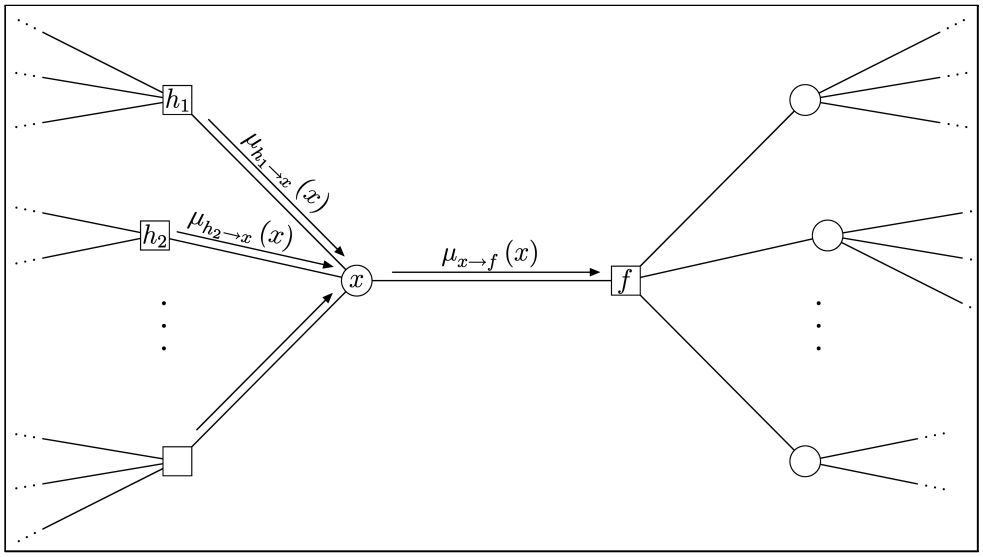

Figura 2.2: Esquema do cálculo de mensagem de um nó-variável para um nó-função no algoritmo SP.

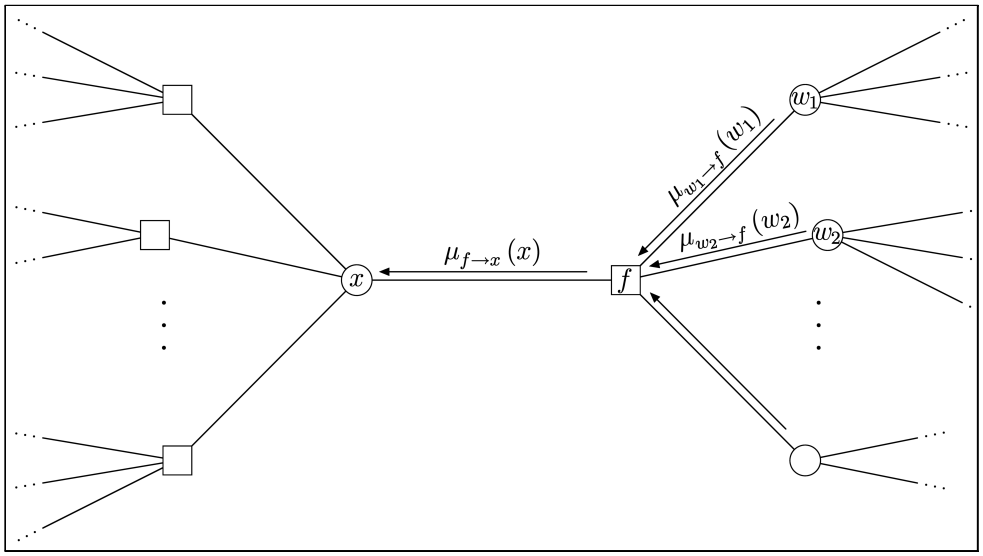

Figura 2.3: Esquema do cálculo de mensagem de um nó-função para um nó-variável no algoritmo SP.

As regras de atualização do algoritmo SP são dadas por

$$\mu_{x\to f}(x) = \prod_{\substack{h\in N(x)\\ h\neq f}} \mu_{h\to x}(x) \tag{2.21}$$

$$\mu_{f\to x}(x) = \sum_{\substack{w\in N(f)\\ w\neq x}} f \prod_{\substack{w\in N(f)\\ w\neq x}} \mu_{w\to f}(w) \tag{2.22}$$

Aqui, $N(x)$ denota o conjunto de vizinhos da variável x no grafo, que são funções, e $N(f)$ são as variáveis vizinhas da função f. Há uma similaridade natural entre as expressões (2.21) e (2.22). A primeira pode ser pensada como um caso particular da segunda, com a função sendo a unidade (elemento neutro multiplicativo), e a marginalização não sendo necessária já que não há variáveis envolvidas além de x.

Ao final do algoritmo, a marginal em cada variável é dada pelo produto de todas as mensagens incidentes em seu nó,

$$\mu(x) = \prod_{f\in N(x)} \mu_{f\to x}(x) \tag{2.23}$$

A partir da expressões gerais (2.21) e (2.22) para cálculo de mensagens, alguns casos particulares podem ser destacados. Como o caso de nós de apenas uma conexão (*leaf nodes*), e portanto apenas um vizinho. No caso de um nó-variável deste tipo, a mensagem que ele envia para sua única conexão é

$$\mu_{x\to f}(x) = 1, \tag{2.24}$$

uma função constante (com o elemento neutro multiplicativo do semi-anel). No caso de um nó-função com apenas uma conexão,

$$\mu_{f\to x}(x) = f \tag{2.25}$$

Em um nó-variável com exatamente dois vizinhos, digamos $N(x) = \{f, h\}$, o nó x apenas repassa as mensagens entre seus nós vizinhos, isto é, $\mu_{x\to f}(x) = \mu_{h\to x}(x)$ e $\mu_{x\to h}(x) = \mu_{f\to x}(x)$.

Para o cálculo de todas as marginais, em todas as variáveis, é necessário que todas as mensagens, em todos os ramos, sejam computadas. Para a obtenção de apenas uma marginal

específica em determinada variável, menos mensagens são requeridas. São as mensagens em cada ramo que seguem na "direção" da variável. De fato, com o grafo disposto de forma centralizada nesta variável, ou com esta variável disposta em destaque num topo do grafo, é possível visualizar quais mensagens seguem em sua direção. Como exemplo usual, podemos explicitar as mensagens envolvidas no cálculo de uma marginalização sobre o exemplo de grafo-fator dado figura 2.1, e seu respectivo re-arranjo com o fluxo de mensagens.

$$\underset{x_1}{\operatorname{marg}}\, g\left(x_1, x_2, x_3, x_4, x_5\right) = \sum_{x_2}\sum_{x_3}\sum_{x_4}\sum_{x_5} f_A\left(x_1\right) f_B\left(x_2\right) f_C\left(x_1, x_2, x_3\right) f_D\left(x_3, x_4\right) f_E\left(x_3, x_5\right) =$$

$$\begin{aligned}
& f_A\left(x_1\right) \quad \sum_{x_2}\sum_{x_3} \underbrace{f_B\left(x_2\right)}\; f_C\left(x_1, x_2, x_3\right) \underbrace{\sum_{x_4} f_D\left(x_3, x_4\right)}\; \underbrace{\sum_{x_5} f_E\left(x_3, x_5\right)} = \\
& f_A\left(x_1\right) \quad \sum_{x_2}\sum_{x_3} \underbrace{\mu_{f_B \to x_2}\left(x_2\right)}\; f_C\left(x_1, x_2, x_3\right) \underbrace{\mu_{f_D \to x_3}\left(x_3\right) \quad \mu_{f_E \to x_3}\left(x_3\right)} = \\
& \underbrace{f_A\left(x_1\right)}\; \underbrace{\sum_{x_2}\sum_{x_3} \mu_{x_2 \to f_C}\left(x_2\right) f_C\left(x_1, x_2, x_3\right) \quad \mu_{x_3 \to f_C}\left(x_3\right)} = \\
& \mu_{f_A \to x_1}\left(x_1\right) \qquad\qquad \mu_{f_C \to x_1}\left(x_1\right)
\end{aligned}$$

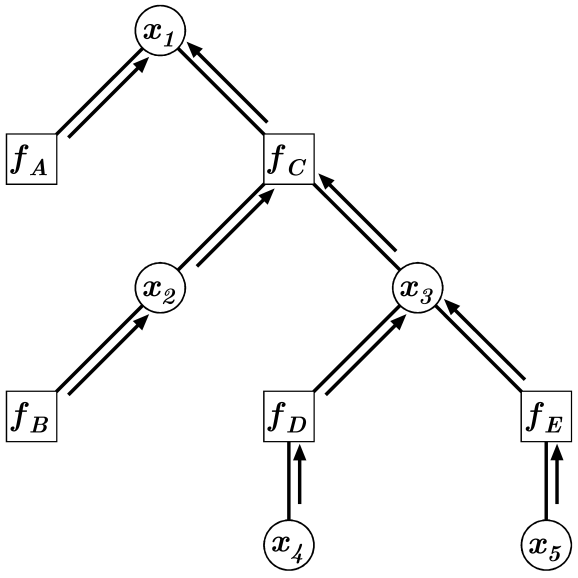

Figura 2.4: Grafo-fator hierarquizado e fluxo de mensagens para cálculo de uma marginal.

Este exemplo mostra o caráter recursivo e distributivo da evolução e geração de mensagens para o algoritmo SP em um grafo sem ciclos. Esta disposição é a chamada representação em árvore (*tree representation*) do grafo, onde a variável com marginal a ser obtida fica no topo, seus primeiros vizinhos uma linha abaixo, e assim por diante. Esse formato evidencia o fato de que num grafo conexo e sem ciclos, fixado um nó referência, fica estabelecida uma ordem parcial entre o restante dos nós. Essa propriedade é central na demonstração do algoritmo SP apresentada a seguir.

Demonstração do algoritmo SP

Num grafo sem ciclos e conexo, fixada uma variável x da qual se deseja extrair a marginal, a idéia é definir várias operações relativas ao nó de x nos outros nós do grafo. De fato, conforme mencionado no exemplo anterior, quando o grafo é disposto na representação em árvore, com x no topo, podemos definir sobre o conjunto dos nós do grafo uma ordem parcial relativa ao nó x, da seguinte forma: $v_1 \succ v_2$ se e só se v_1 pertence ao caminho (único) entre x e v_2. Podemos também dizer que neste caso v_2 é descendente de v_1 relativo a x. No exemplo anterior, em relação a x_1, temos cadeias de descendência do tipo: $x_1 \prec f_C \prec x_3 \prec f_E \prec x_5$ e $x_1 \prec f_C \prec x_2 \prec f_B$. O conjunto de descendentes de um nó v pode ser denotado por $\{u : u \succ v\}$. Seja $N(v)$ o conjunto de descendentes diretos de um elemento, isto é, $N(v) = \{u : u \succ v \ , \ d(u, v) = 1\}$, por simplidade a mesma notação usada para denotar conjunto de vizinhos, embora não necessariamente coincidam.

Para qualquer nó v, o conjunto de seus descendentes junto com ele próprio $\{u : u \succeq v\}$ definem naturalmente um sub-grafo derivado do grafo principal. Para um nó y, seja $\text{marg}(y)$ a marginal relativa ao seu subgrafo associado. Podemos expressar

$$\text{marg}(y) = \sum_{z : z \succ y} \prod_{f : f \succ y} f \quad , \tag{2.26}$$

onde o somatório e o produtório são indexados em nós-variáveis e nós-funções que satisfazem a restrição de pertencerem ao sub-grafo associado a y. Um ponto a destacar é que o operador somatório tem um número variável de indexadores de soma, e que é o conjunto destes indexadores que está denotado na sua parte inferior, e portanto não é um operador somatório no sentido usual. De fato a soma se dá em sub-espaços, produtos cartesianos definidos

anteriormente, que ficam implícitos pela enumeração das variáveis.

Para qualquer função f, as variáveis contidas em seu sub-grafo associado podem ser expressas pela união disjunta

$$N(f) \cup \left(\bigcup_{y \in N(f)} \{z : z \succ y\} \right) \tag{2.27}$$

Para o nó base x, seu sub-grafo associado é o grafo inteiro, e $\operatorname{marg}(x)$ coincide com a marginal usual. Então,

$$\begin{aligned}
\operatorname{marg}(x) &= \sum_{y:y\succ x} \prod_{f:f\succ x} (f) && (2.28)\\
&= \sum_{y:y\succ x} \prod_{f\in N(x)} \left(f \prod_{g:g\succ f} (g) \right) && (2.29)\\
&= \prod_{f\in N(x)} \left(\sum_{y:y\succ f} f \prod_{g:g\succ f} (g) \right) && (2.30)\\
&= \prod_{f\in N(x)} \left(\sum_{y\in N(f)} \sum_{\bigcup_{y\in N(f)} \{z:z\succ y\}} f \prod_{y\in N(f)} \left(\prod_{g:g\succ y} (g) \right) \right) && (2.31)\\
&= \prod_{f\in N(x)} \left(\sum_{y\in N(f)} f \sum_{\bigcup_{y\in N(f)} \{z:z\succ y\}} \left(\prod_{y\in N(f)} \left(\prod_{g:g\succ y} (g) \right) \right) \right) && (2.32)\\
&= \prod_{f\in N(x)} \left(\sum_{y\in N(f)} f \prod_{y\in N(f)} \left(\sum_{z:z\succ y} \prod_{g:g\succ y} (g) \right) \right) && (2.33)\\
&= \prod_{f\in N(x)} \left(\sum_{y\in N(f)} f \prod_{y\in N(f)} (\operatorname{marg}(y)) \right). && (2.34)
\end{aligned}$$

Uma vez que cada y é descendente direto de f na expressão acima, definimos

$$\mu_{y\to f} = \operatorname{marg}(y) \tag{2.35}$$

e uma vez que cada f é descendente direto de x, definimos

$$\mu_{f\to x} = \left(\sum_{y\in N(f)} f \prod_{y\in N(f)} (\operatorname{marg}(y)) \right). \tag{2.36}$$

Assim, obtemos as duas expressões recursivas do algoritmo SP:

$$\mu_{f\to x} = \sum_{y\in N(f)} f \prod_{y\in N(f)} \mu_{y\to f} \quad , \tag{2.37}$$

$$\mu_{y\to f} = \prod_{g\in N(y)} \mu_{g\to y} \quad . \tag{2.38}$$

■

Algumas observações sobre grafos sem ciclos:

- Duas variáveis tem no máximo uma função vínculo entre elas.
- Como cada nó separa o grafo em partes disjuntas (ficando cada um de seus vizinhos em uma parte diferente), há um princípio de independência entre as mensagens que chegam a cada nó por cada ramo. Em particular, há um princípio de independência entre mensagens que passam em direções opostas por dado ramo. Isso vale para grafos sem ciclos, e vale aproximadamente para grafos com ciclos longos.
- Para um dado ramo (x, f), as mensagens que passam nas duas direções são funções da variável x. Pelo princípio da independência no grafo sem ciclos, as mensagens $\mu_{f\to x}$ e $\mu_{x\to f}$ contém informações a partir de lados diferentes do grafo.

O algoritmo SP em grafos sem ciclos, quando realizado completamente, exaurindo todas as mensagens em todos os ramos em ambas as direções, obtém todas marginais simples exatas. Uma implementação para o algoritmo SP baseado no grafo-fator deve considerar um ponto de processamento em cada nó do grafo, um "processador" independente. Cada um destes processadores troca mensagens com seus vizinhos, e cada ramo do grafo é um canal de comunicação entre dois processadores em ambas as direções. Cada nó recebe e envia mensagens com seus vizinhos através destes canais. Cada mensagem não é apenas um valor escalar, mas uma função de uma variável. Se a variável tem domínio finito, a mensagem natural é um vetor com o conjunto de valores da função em cada um dos pontos, numa ordem pré-estabelecida.

Podemos entender o algoritmo SP operando num grafo associado com a seguinte heurística: a marginalização por propagação de mensagens no grafo é feita ao longo dos

ramos. Uma função local passa a ser parte integrante do seu 'nó-função' no grafo, que calcula e transmite mensagens localmente usando esta função custo. Os nós-funções são "caixas pretas" que compartilham variáveis, por uma espécie de "acoplamento". A única informação que cada caixa preta dá é associar um valor (custo) a cada configuração de suas variáveis. No sistema acoplado, variáveis (e seus nós) são objetos passivos, que apenas são influenciadas conjuntamente pelas caixas pretas (diretamente pelas suas vizinhas e indiretamente pelas outras durante a realização do algoritmo) e retransmitem esta influência pela rede de acordo com suas conexões. Cada nó-função é um objeto ativo, com sua função custo influenciando toda a inferência na rede. Por fim, a tarefa do algoritmo é, a partir das funções custos dos nós-funções, obter funções custos resultantes (*a posteriori*) para os nós-variáveis. A partir destas, podem ser extraídas estimativas sobre as melhores configurações de variáveis que otimizam os custos, conjuntamente ou isoladamente.

2.8. Funções locais determinísticas e grafos de Tanner

Quando é possível obter a distribuição conjunta global do sistema a partir de um produto de fatores de funções indicadoras ou distribuições de probabilidades que descrevem os vínculos entre as variáveis, o algoritmo SP pode ser eficientemente aplicado.

Entre as várias modalidades de acoplamentos multiplicativos, dois tipos de funções-nós determinísticas são destaque: os nós de "checagem de paridade" e os nós de "igualdade".

Os nós de "checagem de paridade" são os clássicos, pois nos grafos de Tanner todos os nós são deste tipo. A função associada pode ser definida por

$$f(x_1, \ldots, x_k) = \begin{cases} 1, & \text{se } x_1 + \ldots + x_k = 0; \\ 0, & \text{caso contrário.} \end{cases} \tag{2.39}$$

Neste caso, as operações algébricas são em $\mathbb{Z}_2$. Se denotamos $\delta(x)$ a função delta de Kronecker, então podemos também escrever

$$f(x_1, \ldots, x_k) = \delta(x_1 + \ldots + x_k + 1). \tag{2.40}$$

Os nós de "igualdade" vincula todas as variáveis conectadas a ele, impondo a restrição de que todas tenham valores (ou "estados") iguais. A função associada pode ser definida por

$$f(x_1, \ldots, x_k) = \begin{cases} 1, & \text{se } x_1 = \ldots = x_k; \\ 0, & \text{caso contrário.} \end{cases} \tag{2.41}$$

ou ainda

$$f(x_1, \ldots, x_k) = \delta(x_2 - x_1) \times \ldots \times \delta(x_k - x_{k-1}). \tag{2.42}$$

Esse tipo de nó representa o acoplamento fundamental de variáveis. Em transformações de grafos-fatores, ele também desempenha papel central, pois realiza o que se denomina "clonagem de variável". Esse tipo de transformação é o princípio para se obter grafos-fatores normais [29].

Similares mas não exatamente equivalentes, grafos normais [29] podem ser considerados uma simplificação canônica de grafos-fatores utilizando nós clonadores de variáveis, de tal forma que nós-variáveis tenham grau máximo dois, ou seja, são conectados a no máximo dois nós-funções. Transformações deste tipo não alteram a complexidade essencial do grafo.

O modelo de grafos de Tanner [79] foi concebido como uma generalização gráfica dos códigos LDPC de Gallager, e foi o primeiro a usar a idéia de grafos bipartidos para representar códigos. Grafos-fatores podem ser considerados uma generalização dos grafos de Tanner.

Em particular, os grafos de Tanner para representar códigos de paridade são compostos de duas classes distintas e bem definidas de nós. Uma classe representa todos os símbolos do código, e outra representa todas as checagens de paridade (um nó para cada linha da matriz), e conecta apenas nós entre classes diferentes para representar quais símbolos participam de quais checagens de paridade. O grafo de Tanner define unicamente um código, já que caracteriza completamente a estrutura do código e sua matriz de verificação de paridade. Enquanto cada símbolo é enumerado no grafo por nós rotulados, cada nó de checagem de paridade é representado graficamente sem detalhes adicionais.

Em grafos-fatores genéricos, embora a representação seja similar e inspirada nos grafos de Tanner, os nós conectados aos nós-variáveis são 'funções vínculos' genéricas e não apenas

checagem de paridade. Como consequência, o grafo por si só não mais define completamente o código, apenas a estrutura de conexões.

Por outro lado, códigos mais gerais são naturalmente representáveis quando se considera grafos-fatores. Códigos de treliça são representados por funções locais determinísticas associadas às seções da treliça.

2.9. Realização generalizada por estados de um código

É comum na literatura existente uma distinção entre dois tipos de variáveis em grafos-fatores: "variáveis símbolos" e "variáveis ocultas". Na descrição de códigos de treliça, símbolos de entrada e saída num codificador são do primeiro tipo, e a sequência de variáveis que representa a evolução dos estados do codificador são do segundo tipo.

Em [29], Forney formaliza o conceito de "realização generalizada por estados", com os três elementos: um conjunto de variáveis símbolos, um conjunto de variáveis de estados, e um conjunto de vínculos. O código é obtido pela "projeção" no sub-espaço de variáveis símbolos, ou ainda pelo "puncionamento" das variáveis de estados.

Algumas características na interpretação de um sistema típico: variáveis símbolos são dadas *a priori*, fundamentais e inerentes ao código em questão; variáveis ocultas são introduzidas adicionalmente para uma descrição alternativa para o mesmo código, uma realização. Portanto, o número de variáveis ocultas e o papel de cada uma pode mudar dependendo da realização do código.

Essa distinção não é realmente necessária, já que no algoritmo SP e marginalização todas as variáveis são tratadas da mesma forma.

De fato, esta diferenciação é útil para definir algumas variáveis que são auxiliares na descrição do sistema, e serão excluídas por marginalização de todos os cálculos finais, e suas marginais próprias não são de interesse para o problema. Em última instância, o objetivo do uso de variáveis auxiliares é simplificar as funções vínculos necessárias para descrever um sistema, ou seja, possibilitar uma fatoração mais analítica ao sistema, com

mais desmembramentos e independências parciais entre as variáveis de interesse. O caso típico desta situação é a descrição de códigos por treliças e a introdução de variáveis de estado como auxiliares.

Podemos estabelecer então que duas funções globais de um mesmo sistema

$$f\left(u_1, \ldots, u_k, c_1, \ldots, c_n\right) \tag{2.43}$$

e

$$f'\left(u_1, \ldots, u_k, c_1, \ldots, c_n, s_1 \ldots, s_n\right) \tag{2.44}$$

são consideradas equivalentes para o código se a marginalização da segunda resulta na primeira, isto é,

$$f\left(u_1, \ldots, u_k, c_1, \ldots, c_n\right) = \sum_{s_1 \ldots, s_n} f'\left(u_1, \ldots, u_k, c_1, \ldots, c_n, s_1 \ldots, s_n\right). \tag{2.45}$$

Neste caso, podemos pensar f' como uma extensão de f. O caso de interesse é quando f' envolve mais variáveis, mas permite uma fatoração mais analítica. As duas funções f e f' podem descrever o mesmo sistema relativo às variáveis $\{u_1, \ldots, u_k, c_1, \ldots, c_n\}$ desde que sua marginalização resulte na mesma função para estas.

2.10. Grafo-fator típico para um sistema de comunicação

A estrutura clássica de um sistema de comunicação - fonte, codificação e canal - pode ser sumariamente descrita no contexto da teoria de grafos-fatores. As variáveis caraterísticas são

- símbolos (bits) de informação $\mathbf{u}$;
- os estados do codificador $\mathbf{s}$;
- símbolos codificados transmitidos $\mathbf{c}$;

- e os sinais recebidos do canal $\mathbf{y}$,

onde $\mathbf{u}$, $\mathbf{s}$, $\mathbf{c}$ e $\mathbf{y}$ são vetores, representações compactas das sequências de valores. Em geral, estas variáveis e suas inter-relações definem completamente a estrutura de codificação e decodificação para o sistema dado. Essas relações sempre podem ser expressas probabilisticamente, pelas funções distribuições inerentes a cada parte do sistema. De fato, em todos os sistemas típicos, podemos escrever a distribuição conjunta como

$$P(\mathbf{u},\mathbf{s},\mathbf{c},\mathbf{y}) = P(\mathbf{u})\,P(\mathbf{s}|\mathbf{u})\,P(\mathbf{c}|\mathbf{u},\mathbf{s})\,P(\mathbf{y}|\mathbf{c})\,. \tag{2.46}$$

Tipicamente, $P(\mathbf{u})$ é composta de distribuições uniformes, $P(\mathbf{s}|\mathbf{u})$ e $P(\mathbf{c}|\mathbf{u},\mathbf{s})$ representam relações determinísticas, e estes componentes se fatoram em sequências de distribuições locais. $P(\mathbf{y}|\mathbf{c})$ expressa o modelo do canal e também se fatora em componentes. O caso mais simples é quando não há memória no modelo do canal.

Podemos ainda completar a representação (2.46) incluindo a observação das variáveis do canal como componente intrínseco ao problema de decodificação. De fato, este é mais um fator acoplado à distribuição conjunta. Em geral, a representação funcional para observação de variáveis são deltas de Dirac ou deltas de Kronecker. Portanto, a distribuição conjunta do problema completo de decodificação é da forma

$$P(\mathbf{u},\mathbf{s},\mathbf{c},\mathbf{y}) = P(\mathbf{u})\,P(\mathbf{s}|\mathbf{u})\,P(\mathbf{c}|\mathbf{u},\mathbf{s})\,P(\mathbf{y}|\mathbf{c})\,\delta(\mathbf{y})\,. \tag{2.47}$$

É comum na literatura manipular os cálculos de decodificação pensando nos valores observados como 'parâmetros', ou variáveis com valores fixados. Entretanto, numa representação funcional do sistema, a forma (2.47) acima fica mais coerente. É interessante pensar que a observação de variáveis ainda mantém a forma geral de uma fatoração com funções em todas as variáveis do sistema, apenas acoplando um fator adicional, e o problema ainda se resume à obtenção de marginais desta conjunta, e não uma conjunta 'parametrizada' por algumas variáveis. Isso evidentemente não simplifica cálculos nem tem efeito prático relevante, mas conserva um entendimento unificado sobre a caracterização de variáveis e funções no grafo.

Assim, temos como resultado uma fatoração que expressa todos os acoplamentos envolvidos entre os tipos de variáveis do sistema. O fato de termos estes componentes com

separação bem definida na fatoração da conjunta também indica que a decodificação pode ser realizada por etapas em cada um destes componentes.

Abstraindo o fato de serem distribuições de probabilidade, podemos expressar a fatoração (2.47) acima como

$$f(\mathbf{u},\mathbf{s},\mathbf{c},\mathbf{y}) = f_{priori}(\mathbf{u})\, f_{evol}(\mathbf{u},\mathbf{s})\, f_{cod}(\mathbf{u},\mathbf{s},\mathbf{c})\, f_{canal}(\mathbf{c},\mathbf{y})\, f_{observ}(\mathbf{y}). \tag{2.48}$$

e o seu respectivo grafo-fator

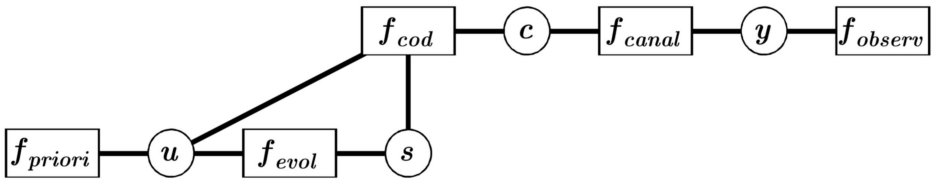

Figura 2.5: Grafo-fator de um sistema com um codificador de estados.

Desta forma, a teoria de grafos-fatores e seu algoritmo distributivo generalizado aplicado a este caso leva a entender uma decodificação como a marginalização de uma medida global do sistema, obtida pela junção das medidas dos componentes. E todos os componentes típicos de um sistema de comunicação já tem naturalmente sua medida definida em sua descrição, probabilística ou comportamental, acopladas multiplicativamente (numa álgebra apropriada) tal que o algoritmo SP se aplica. Variáveis observadas e variáveis ocultas são explícita ou implicitamente marginalizadas no processo.

Numa visão mais geral, num grafo-fator, se duas funções f_{j_1} e f_{j_2} tem x_i como argumento comum, então f_{j_1} e f_{j_2} estão acopladas através de x_i.

Num sistema com variáveis aleatórias X, Y e Z temos que $X \perp Y|Z$ (X e Y são independentes dado Z) se e somente se a distribuição conjunta pode ser escrita fatorada

$$p(x,y,z) = f(x,z)\, g(y,z)\ , \tag{2.49}$$

isto é, se tal fatoração existe. Aqui, temos que f e g estão acopladas através de z.

Os modelos gráficos para sistemas de variáveis aleatórias essencialmente tem como objetivo descrever a estrutura de dependência (e independência) entre elas, de forma que a distribuição conjunta seja fatorada em distribuições condicionais mais simples, com poucas variáveis, o que resulta numa caracterização local para cada uma delas no grafo. A teoria de grafos-fatores generaliza esta idéia, com abrangência maior do que apenas fatorações de distribuições de probabilidade, embora em última instância todos os casos práticos mais comuns possam ainda ser reduzidos à descrição probabilística.

O processo de decodificação com o critério de decisão MAP (*Maximum-A-Posteriori*) pode ser simplesmente escrito como

$$\begin{aligned} \hat{u} &= \underset{u}{\operatorname{argmax}}\, p\left(u|y=y_0\right) \\ &= \underset{u}{\operatorname{argmax}} \left(\frac{p\left(u,y\right)|_{y=y_0}}{p\left(y\right)|_{y=y_0}} \right) \\ &= \underset{u}{\operatorname{argmax}} \left(p\left(u,y\right)|_{y=y_0} \right) \\ &= \underset{u}{\operatorname{argmax}} \sum_{\tilde{u}} \left(p\left(u,y\right) \delta_{y_0}\left(y\right) \right) \ , \end{aligned} \tag{2.50}$$

onde fica evidente que a operação $\underset{u}{\operatorname{argmax}}$ é realizada posteriormente à operação de marginalização na variável u, quando considerada uma função global que inclui como fator uma função delta para a variável observada. Na expressão (2.50) acima, $\delta_{y_0}\left(y\right) \triangleq \delta\left(y-y_0\right)$. Assim, a decisão MAP pode ser entendida como a marginalização de uma função global em todas as variáveis menos u. Cada variável observada implica no acoplamento na conjunta de uma função delta como fator.

2.11. Grafo-fator de uma máquina de estados

Nesta seção é incluída uma descrição mais detalhada do grafo-fator associado a um codificador de estados (convolucional), ou mais genericamente, a um código de treliça.

Sejam $(U_1, U_2, \ldots)$ as variáveis aleatórias que representam a sequência de símbolos de

entrada, não codificados. As variáveis $(S_0, S_1, \ldots)$ são a sequência de estados do codificador. A sequência de símbolos de saída, os símbolos codificados, são representados por $(C_1, C_2, \ldots)$, e $(Y_1, Y_2, \ldots)$ são as variáveis observadas *a posteriori*, que no caso são versões ruidosas de $(C_1, C_2, \ldots)$. Para uma sequência curta podemos expressar a fatoração completa e o respectivo gráfico na figura 2.6 a seguir

$$\begin{aligned}\Pr\{u_0, u_1, s_0, s_1, s_2, c_0, c_1, y_0, y_1\} = \\ \Pr\{u_0\}\Pr\{u_1\}\Pr\{s_0\}\Pr\{c_0, s_1|s_0, u_0\}\Pr\{c_1, s_2|s_1, u_1\}\Pr\{y_0|c_0\}\Pr\{y_1|c_1\}\end{aligned} \tag{2.51}$$

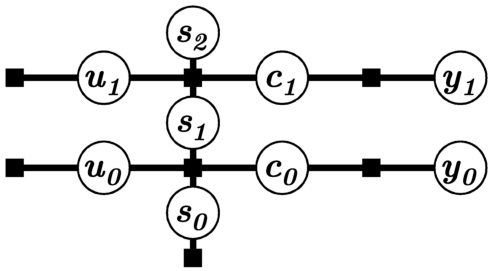

Figura 2.6: Grafo-fator de uma máquina de estados.

Neste caso, as funções do tipo $\Pr\{c_t, s_{t+1}|s_t, u_t\}$ são funções indicadoras que descrevem as associações na seção da treliça, e ainda admitem fatoração adicional do tipo

$$\Pr\{c_t, s_{t+1}|s_t, u_t\} = \Pr\{c_t|s_t, u_t\}\Pr\{s_{t+1}|s_t, u_t\} \tag{2.52}$$

As probabilidades $\Pr\{s_{t+1}|s_t, u_t\}$ definem a evolução dos estados, e $\Pr\{c_t|s_t, u_t\}$ definem o símbolo emitido em função do ramo da treliça.

Um entendimento já estabelecido na literatura [54], o algoritmo SP aplicado ao grafo do código de treliça resulta no algoritmo *forward-backward* (ou BCJR, dado em [4]).

Mas além disso, é possível neste caso estabelecer significado probabilístico preciso para cada uma das mensagens que fluem pelo grafo, conforme a figura 2.7 a seguir, onde sequências de variáveis são denotadas abreviadamente como $y_s^t \triangleq y_s, \ldots, y_t$, e aqui estas variáveis observadas devem ser consideradas fixas, e tomadas como parâmetros.

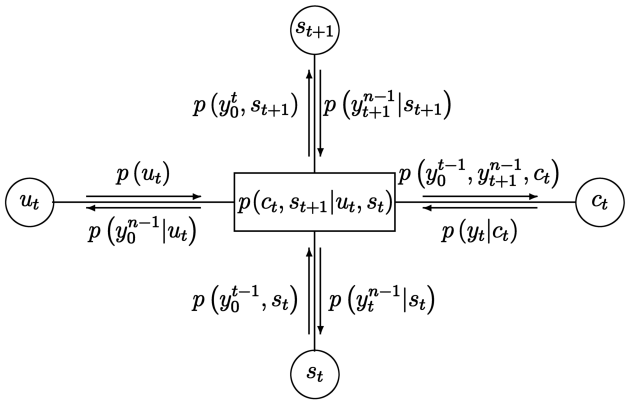

Figura 2.7: Mensagens do algoritmo SP para uma máquina de estados.

Por simplificação, consideramos que as várias funções, todas denotadas por $p\left(\cdot\right)$ e variados argumentos, são distribuições de probabilidades nestes argumentos. Lembrando que as probabilidades $p\left(u_t\right)$ e $p\left(y_t|c_t\right)$, $t = 0, \ldots, n-1$ podem ser consideradas vetores de alimentação para o algoritmo BCJR.

Temos um grafo sem ciclos, e cada nó-variável tem no máximo duas conexões com nós-funções. Podemos identificar a independência de informação fluindo em direções opostas pelos pares de mensagens. As mensagens $p\left(y_0^{t-1}, s_t\right)$ e $p\left(y_t^{n-1}|s_t\right)$ referentes ao nó-variável S_t (e portanto funções simples no argumento s_t) incluem todas as observações $y_0, \ldots, y_{n-1}$ como parâmetros, uma parte delas em cada distribuição. Essa partição das observações obedece um princípio de causalidade, ou seja, passado e futuro em relação ao instante t. Os dois conjuntos de variáveis $Y_0, \ldots, Y_{t-1}$ e $Y_t, \ldots, Y_{n-1}$ são independentes e complementares. A marginal na variável S_t é dada pelo produto das duas mensagens,

$$p\left(y_0^{n-1}, s_t\right) = p\left(y_0^{t-1}, s_t\right) p\left(y_t^{n-1}|s_t\right). \tag{2.53}$$

Aqui, os três elementos devem ser considerados funções em uma variável (s_t) e $y_0, \ldots, y_{n-1}$ como parâmetros. Assim, a marginal consiste na conjunta da variável s_t com todas as variáveis observadas, com seus respectivos valores $y_0, \ldots, y_{n-1}$ observados.

Considerando o particionamento do grafo pelo nó S_t, note que as duas mensagens carregam a informação relevante (observações *a posteriori*) de cada um dos dois sub-grafos

resultantes.

Interpretação idêntica pode ser feita para as mensagens referentes ao nó S_{t+1}, similares a S_t a menos de uma translação.

Analogamente, $p\left(y_0^{t-1}, y_{t+1}^{n-1}, c_t\right)$ e $p\left(y_t|c_t\right)$ referentes ao nó C_t também apresentam particionamento dos parâmetros observados em relação aos sub-grafos que ele separa, e a marginal tem seu significado preciso

$$p\left(y_0^{n-1}, c_t\right) = p\left(y_0^{t-1}, y_{t+1}^{n-1}, c_t\right) p\left(y_t|c_t\right). \tag{2.54}$$

E finalmente, nas variáveis U_t, temos o par de mensagens $p\left(u_t\right)$ e $p\left(y_0^{n-1}|u_t\right)$, uma partição trivial dos parâmetros (todos concentrados em uma delas). Temos $p\left(u_t\right)$ e $p\left(y_0^{n-1}|u_t\right)$ como mensagens independentes e complementares, e a marginal

$$p\left(y_0^{n-1}, u_t\right) = p\left(u_t\right) p\left(y_0^{n-1}|u_t\right). \tag{2.55}$$

Portanto, o exemplo clássico do grafo-fator de um código de treliça permite uma completa e precisa interpretação sobre o significado de todas as mensagens que fluem através dele no algoritmo SP. Através de cada variável, as duas mensagens associadas são distribuições, uma delas na qual a variável aparece condicionada, e outra não. E todas as observações $y_0, \ldots, y_{n-1}$ participam deste par de mensagens como parâmetros, particionados de acordo com os sub-grafos associados.

Essa similaridade induz uma caracterização das mensagens que fluem no grafo, separando-as em duas classes de mensagens distintas, com respeito a causalidade. Mensagens do tipo *forward* , denotadas $\alpha\left(u_t\right)$, $\alpha\left(s_t\right)$, $\alpha\left(c_t\right)$ e mensagens do tipo *backward* , $\beta\left(u_t\right)$, $\beta\left(s_t\right)$, $\beta\left(c_t\right)$. As primeiras fluem na direção da causalidade entre as variáveis, e as segundas, na direção oposta. Desta forma, as expressões do algoritmo *forward-backward* podem ser apresentadas

(com espaçamento nos produtos para alinhar mensagens) de forma completa

$$\begin{aligned}
\alpha(s_{t+1}) &= \sum_{u_t,s_t,c_t} T(s_t,u_t,c_t,s_{t+1})\ \alpha(u_t)\ \alpha(s_t)\ \beta(c_t) \\
\beta(s_t) &= \sum_{u_t,c_t,s_{t+1}} T(s_t,u_t,c_t,s_{t+1})\ \alpha(u_t)\ \beta(c_t)\ \beta(s_{t+1}) \\
\beta(u_t) &= \sum_{s_t,c_t,s_{t+1}} T(s_t,u_t,c_t,s_{t+1})\ \alpha(s_t)\ \beta(c_t)\ \beta(s_{t+1}) \\
\alpha(c_t) &= \sum_{u_t,s_t,s_{t+1}} T(s_t,u_t,c_t,s_{t+1})\ \alpha(u_t)\ \alpha(s_t)\ \beta(s_{t+1})
\end{aligned} \tag{2.56}$$

onde

$$T(s_t,u_t,c_t,s_{t+1}) \triangleq \Pr\{c_t,s_{t+1}|s_t,u_t\}, \tag{2.57}$$

$$\alpha(u_t) \triangleq \Pr\{u_t\}, \tag{2.58}$$

$$\beta(c_t) \triangleq \Pr\{y_t|c_t\} \tag{2.59}$$

são os elementos primários, a partir dos quais o algoritmo é executado, além das mensagens iniciais nas extremidades da treliça $\alpha(s_0)$ e $\beta(s_{n-1})$, que ficam em função dos critérios de início e término da mesma na codificação. Para valores extremos fixos e conhecidos, estas mensagens são do tipo deltas de Kronecker $\delta(\cdot)$. Para treliças truncadas onde a extremidade tem estado desconhecido, são mensagens do tipo constante em todo o domínio.

Podemos destacar aqui que a última das quatro fórmulas em (2.56), para $\alpha(c_t)$, não é usualmente apresentada pela literatura, mas é parte integrante do algoritmo em sua plenitude. Ela é efetiva em concatenações turbo seriais.

2.12. Códigos concatenados e grafos com ciclos

Em geral, as equações que definem as regras de cálculos das mensagens no algoritmo SP dadas por (2.21) e (2.22) são recursivas e localmente aplicáveis a qualquer grafo-fator, pois só envolvem variáveis e funções em uma vizinhança. Analogamente, a regra de obtenção de marginais, dada por (2.23), só envolve mensagens locais. São estes fatos que possibilitam a extensão do algoritmo SP para grafos com ciclos. Entretanto, em um grafo com ciclos, a aplicação do algoritmo SP resulta naturalmente em um algoritmo iterativo. Uma completa

implementação deve definir um critério de inicialização, uma ordem de propagação de mensagens (*scheduling*), e um critério de parada.

Para um grafo sem ciclos, há uma ordem natural. Se o problema se resume a obter uma única marginal, os cálculos se iniciam pelos nós mais distantes da variável objetivada, e seguem em sua direção. Para o problema completo de se obter todas as marginais, o algoritmo possibilita cálculos simultâneos (processamento paralelo) e eficientes em vizinhanças independentes no grafo. O grafo pode assumir uma disposição de máximo espalhamento entre os nós, tal que haja equilíbrio de ramificações para todos os lados, evidenciando o distanciamento de cada nó em relação a um ponto central. Os cálculos começam pelos nós periféricos da árvore, e as mensagens evoluem, um fluxo que inicia centrípeto, gradualmente centraliza mensagens na sua primeira metade, e termina centrífugo na sua segunda metade, terminando nas mensagens que chegam a nós mais periféricos.

Já em grafos com ciclos, a realização do algoritmo para um dado grafo requer a arbitragem de uma regra adicional, que defina a ordem na qual os cálculos são efetuados, o cronograma de execução (*scheduling*). Para o caso dos códigos turbo e LDPC, o cronograma herdado diretamente de seus esquemas clássicos evidencia a equivalência entre a decodificação iterativa usual destes algoritmos e a decodificação pelo algoritmo SP.

Qualquer código, quando realizado seu grafo-fator, pode ser decodificado via algoritmo SP. Mas o algoritmo SP pode ser provado exato apenas para grafos sem ciclos, onde temos critérios de otimalidade satisfeitos em diferentes instâncias do algoritmo, MAP ou MV (Máxima Verossimilhança).

Num grafo sem ciclos, as marginais obtidas por (2.23) podem ser usadas para decisões MAP símbolo a símbolo. Num grafo com ciclos, o máximo que pode se esperar é que depois de algumas iterações as "marginais" obtidas por (2.23) sejam boas aproximações das marginais reais, o que resulta em uma decodificação MAP aproximada.

Realizações de códigos por grafos com ciclos permitem construção, descrição e implementação simples de códigos que são complexos nas suas realizações sem ciclos. Portanto, tanto a introdução de variáveis-estado auxiliares quanto a introdução deliberada de ciclos

na construção de realizações de códigos sobre grafos podem ser pensadas como técnicas de se construir códigos potentes (complexos na sua descrição clássica), mas com baixa complexidade de implementação. Assim, para cada realização de um mesmo código, a complexidade de decodificação varia. E para cada realização com ciclos, a performance do algoritmo SP também pode variar.

O algoritmo SP e suas instâncias fornecem como resultado marginais ou pseudo marginais para variáveis (o que Wiberg chamou de funções custo finais [85]), e não decisões finais sobre valores. Portanto, segue depois a etapa de como as decisões ou estimativas a partir destas marginais serão obtidas, que depende de cada tipo de problema.

2.13. Notas históricas e considerações adicionais sobre grafos-fatores

Parte da literatura se refere a grafos-fatores como grafos de Tanner&Wiberg. Embora tenha sido apresentada em um texto definitivo em 2001 por Kschischang, Frey e Loeliger [54], um artigo tutorial que se tornou principal referência na área, os principais conceitos desta teoria já estavam presentes na literatura anterior, devido a estes e outros pesquisadores. A nomenclatura "grafo-fator" (factor graph) apareceu pela primeira vez em 1998 num artigo de Kschischang e Frey [53]. Os principais autores que contribuíram para o desenvolvimento dos conceitos na área foram Tanner, Wiberg, Frey, Kschischang, Loeliger, Kötter, Aji, Forney, e McEliece.

O ponto inicial no desenvolvimento de códigos definidos sobre grafos veio da grande invenção de Gallager no início dos anos 60 [31] [32], a classe de códigos definidos por matrizes de paridades extensas e esparsas, os códigos LDPC (*Low Density Parity Check*), e seu algoritmo de decodificação associado, uma decodificação iterativa baseada em probabilidades *a posteriori* realizada símbolo a símbolo. A grande relevância dos conceitos desta teoria foi percebida nos anos 90, quando se mostrou por simulações numéricas [59] que de fato se tratava de uma classe importante de códigos, e que muito antes da revolução dos códigos turbo já

havia códigos factíveis com desempenhos próximos à capacidade de canal. Os trabalhos de Sipser e MacKay ([77] e [59]) foram os responsáveis pela redescoberta dos códigos LDPC.

Historicamente, a primeira instância do algoritmo SP para decodificação foi formulada por Gallager [31] [32], e os códigos LDPC podem ser considerados os primeiros códigos definidos sobre grafos. Tanner [79], com base nestes códigos, em 1981 fundou a área de códigos sobre grafos. Tanner introduziu grafos bipartidos para descrever códigos, e definiu uma generalização dos códigos LDPC, com vínculos descritos por funções genéricas em vez de simples checagem de paridades, mostrando como classes de códigos já existentes até então podiam ser representadas desta forma, e que todos eles admitiam a decodificação probabilística por símbolos do SP.

Tanner ao definir seus grafos considerou apenas vínculos determinísticos, ou "comportamentais", e sem variáveis de estados. Sobre o *framework* mais geral de grafos-fatores, um grafo de Tanner é um grafo-fator de alguma fatoração em funções características para o código.

Wiberg apresentou em sua tese [85] uma visão unificada sobre alguns dos mais conhecidos algoritmos de decodificação iterativa, através do algoritmo SP e a representação por grafos dos códigos. Até algoritmos não iterativos, como o de Viterbi, são incluídos nesta visão unificada, mostrando que todos estes são casos particulares do algoritmo SP.

Se Tanner generalizou o papel dos nós-funções, Wiberg generalizou o papel dos nós-variáveis, introduzindo o conceito de "variáveis estados", e mostrando que códigos de treliça também seriam naturalmente representados pelo mesmo tipo de grafo, o que estabeleceu a conexão com a teoria de códigos sobre treliças, o arcabouço teórico convencional de realizações de códigos por modelo de estados. Desta forma, grafos passaram a conter não só as variáveis correspondentes aos símbolos do código, mas também variáveis auxiliares generalizadas, e sistemas podendo ser descritos por várias realizações de grafos equivalentes.

Paralelamente, em outra área de pesquisa, em 1983, Pearl apresentou o algoritmo BP (*belief propagation*) [52] [68], um algoritmo de passagem de mensagens (propagação) a partir da representação em modelos gráficos de uma dada distribuição conjunta, para obter

distribuições marginais, ou seja, para resolver de forma geral e sistemática o chamado *marginalisation problem*.

Inserido inicialmente no contexto de inteligência artificial, com sua generalidade e versatilidade, o algoritmo BP mostrou-se útil em diversas áreas do conhecimento, como na física, biologia, estatística, e teoria de códigos corretores de erro. Entretanto, a essência deste algoritmo já estava presente no algoritmo de decodificação de Gallager [32].

Em [64] a decodificação turbo foi relacionada à esta teoria, sendo um caso particular do algoritmo BP, com o algoritmo de propagação de probabilidades que opera na rede bayesiana estabelecida para o código.

A estreita relação entre o algoritmo BP e alguns algoritmos de decodificação foi apresentada pela primeira vez em 1995 por MacKay [60], que redescobriu os códigos LDPC de Gallager e mostrou que sua decodificação iterativa é um caso particular do algoritmo BP. Assim, todas estas interseções de áreas do conhecimento sugeriam uma unificação, que veio com a teoria de grafos-fatores.

Casos particulares do algoritmo que hoje chamamos genericamente SP foram desenvolvidos ao longo da história em diversas áreas do conhecimento. A primeira instância do SP trazida por Gallager e seus códigos LDPC no início dos anos 60 já indicava a necessidade da inexistência de ciclos para uma marginalização exata. Na mesma época, surgiu o primeiro algoritmo *forward-backward* que se tem notícia, o algoritmo Baum-Welch. Posteriormente, surgiu o algoritmo de Viterbi em 1967 e o algoritmo BCJR em 1974. O algoritmo BP de Pearl foi apresentado em 1983. E finalmente, o algoritmo de decodificação turbo em 1993.

Há algumas alternativas similares ao *framework* de grafos-fatores. A representação por *junction trees* de uma função global [3] traz essencialmente os mesmos resultados, mas as derivações não são tão intuitivas ao não associar elementos gráficos a funções-fatores. Grafos normais [29] podem ser considerados uma simplificação canônica de grafos-fatores, de tal forma que nós-variáveis tenham grau máximo dois, ou seja, são conectados a no máximo dois nós-funções.

Capítulo 3

Grafos-fatores e decodificação turbo

O objetivo deste capítulo é relacionar a teoria geral de grafos-fatores apresentada no capítulo anterior com a decodificação turbo iterativa, algumas de suas variantes e generalizações. Em especial, apresentar propostas de cronogramas de cálculo de mensagens do algoritmo SP de forma mais abrangente, com ênfase no caso do paradigma turbo *stream-oriented*. As principais referências utilizadas foram [7], [8], [9], [35], [42], [43] e [46].

Conforme mencionado no capítulo anterior, a literatura existente em grafos-fatores já identificou o algoritmo turbo como um caso particular do algoritmo SP. Entretanto, o nosso desenvolvimento detalha melhor esta inserção, com contribuições úteis a um entendimento melhor das questões particulares envolvidas nesta instância do SP.

O ponto de partida é o esquema turbo tradicional, definindo seu grafo-fator completo para decodificação por marginalizações, genérico em relação à indexação das sequências de variáveis características do sistema, em uma realização e esquematização que se preocupam em conservar a noção de causalidade entre todas as variáveis envolvidas. Surge daí uma derivação específica na caracterização das mensagens do algoritmo SP.

A teoria de códigos turbo tornou-se uma importante área de pesquisa, na qual há inúmeras análises e desdobramentos. Aqui, focalizamos na análise de sua decodificação no contexto de grafos-fatores. Em relação ao projeto de codificação, incluídos no texto estão apenas definições e conceitos mínimos para o entendimento do que se segue.

3.1. Introdução

Técnicas de decodificação iterativa - como a apresentada por Gallager para LDPC, e a decodificação turbo - são as mais consagradas alternativas para projetar sistemas corretores de erros de alto desempenho, em alguns casos próximos à capacidade de canal, com uma complexidade que cresce linearmente em função do comprimento de bloco. Quando estabelecidos os grafos-fatores para estes códigos, observa-se que estas técnicas de decodificação coincidem localmente com o algoritmo SP.

Sistemas de códigos concatenados consistem em uma coleção de códigos que interagem entre si, cada um deles com complexidade aceitável para codificação e decodificação. O sistema como um todo corresponde a um código global equivalente mais complexo.

Em sistemas de códigos concatenados com decodificação iterativa, como turbo e similares, é comum cada código constituinte ou componente independente ter um grafo sem ciclos. Esses componentes são acoplados e passam a compartilhar variáveis, o que resulta num grafo global com ciclos. Para cada componente, o algoritmo resultaria numa decodificação 'exata', e as marginais obtidas seriam as probabilidades símbolo a símbolo *a posteriori* dos símbolos observados, prontas para decisão ótima MAP, se o componente estivesse isolado. No acoplamento, a propagação de probabilidades resulta num algoritmo aproximado para decodificação MAP. Se o sistema for projetado corretamente, com ciclos longos e aleatórios, temos convergência e bom desempenho.

A decodificação em geral segue pela aplicação do algoritmo internamente a cada código componente e suas variáveis. Probabilidades obtidas em um dado componente são propagadas, geralmente numa ordem pré-definida e cíclica, para o próximo componente. Essa ordem, traduzida para o contexto de grafos-fatores e o algoritmo SP, resulta no cronograma de atualização de mensagens.

3.2. A codificação turbo

Códigos turbo são eficazes códigos corretores de erro para transmissão em canais ruidosos, com razoável complexidade de implementação e decodificação, cuja descoberta foi apresentada em 1993 [9], com desempenho próximo aos limites de Shannon para a capacidade de canal nunca antes obtida, graças à sua até então inovadora técnica de decodificação iterativa e distribuída. Também são flexíveis em termos de comprimento de bloco e taxa de codificação.

Essencialmente, o código turbo padrão consiste numa concatenação paralela de códigos convolucionais, onde cada um deles recebe como entrada a sequência de bits não codificados em diferentes ordens por entrelaçamento. A cada código convolucional componente é associado um decodificador de treliça com algoritmo BCJR MAP, e a decodificação é feita iterando as estimativas em cada decodificador componente.

Generalizando, um sistema de codificação é dito turbo (ou que utiliza o princípio turbo) se ele consiste numa concatenação de códigos de treliça, com suficientes entrelaçadores acoplados entre eles para "quebrar memória relativa", e que a decodificação é iterativa e distribuída entre componentes decodificadores correspondentes, utilizando o algoritmo BCJR MAP ou uma versão dele em cada componente. Uma visão geral sobre concatenações turbo quaisquer é dada em [8], com construções abstratas de redes de códigos.

Podemos ainda falar em uma "classe turbo de códigos" para englobar toda a classe de códigos com algoritmos de decodificação iterativa distribuída símbolo a símbolo. Assim estão incluídos nesta nomenclatura: concatenações paralelas, concatenações seriais, códigos *repeat accumulate* (RA) e códigos de Gallager (LDPC), com todas as variações que estes sistemas possibilitam.

Como tópicos de estudo na vasta literatura a respeito de códigos turbo, podemos citar alguns como: análise estrutural do código equivalente e distância mínima, análise em função do comprimento de bloco, limitantes de desempenho, projetos de entrelaçadores aleatórios e determinísticos, diferenças de desempenho entre concatenação paralela e concatenação serial, desempenho em função da memória nos códigos convolucionais e melhores geradores

recursivos de paridade, diferentes métodos de terminação nas treliças em cada bloco nos codificadores, diferentes métodos numéricos para o algoritmo MAP e suas aproximações, estudo sobre iterações na decodificação e critérios de parada, análise da convergência, etc.

3.3. O codificador turbo clássico

O esquema de uma codificação turbo, em sua concepção original, é esquematizado na figura 3.1 na forma de diagrama de blocos (conforme a abordagem de [8] para construção de redes de códigos). Cada bloco do diagrama corresponde a um componente fundamental.

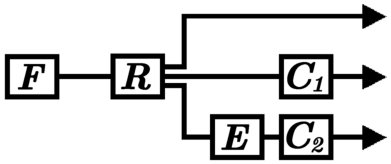

Figura 3.1: Diagrama de blocos de uma codificação turbo.

Temos representados uma fonte binária F com probabilidade uniforme de emissão de bits (não é usual incluir um bloco para a fonte no esquema, mas aqui optamos por fazê-lo já antecipando um modelo mais avançado onde a decodificação envolve um fonte não trivial), acoplada a um codificador turbo, que consiste em: um repetidor R (nomeado *broadcaster* em [8]), geradores convolucionais de paridade C_1 e C_2, e um entrelaçador E conectado à entrada de C_2. Os símbolos binários emitidos em cada um dos três segmentos são serializados e emitidos para um mesmo canal, usualmente modelado como sendo um canal gaussiano com sinalização binária, o canal B-AWGN (Binary Additive White Gaussian Noise).

A decodificação turbo é iterativa e distribuída, usando os chamados blocos SISO (*soft-input/soft-output*) que processam e trocam estas mensagens, as decisões suaves (*soft*). Uma completa descrição de tais blocos é dada em [8].

O sistema de decodificação é dual ao sistema de codificação, onde cada codificador componente tem um decodificador componente associado, e estes componentes trocam

informação iterativamente. Cada um faz sua inferência sobre bits da sequência de informação e passa aos outros. As informações que fluem pelos componentes tem caráter probabilístico (ou de níveis de confiabilidade), não precipitando decisões antes do término da decodificação. Neste sentido, diz-se que a decodificação turbo lida com informação *soft*, em vez de decisões *hard* usadas em outros esquemas de correção de erro.

Os decodificadores associados a C_1 e C_2 implementam o algoritmo MAP na sequência da treliça, tomam como dados iniciais as observações ruidosas dos provenientes do canal, e a cada iteração fornecem como saída "informações extrínsecas" de todos os seus bits de entrada no bloco. A decodificação da sequência de estados de cada código componente é baseado no algoritmo de decodificação MAP sequencial, classicamente denominado BCJR [4]. A decodificação iterativa não é uma decodificação MAP exata do conjunto global, mas empiricamente evidencia desempenho sub-ótimo, próximos aos limites teóricos de capacidade, quando o sistema é projetado corretamente.

3.3.1. Geradores de paridade convolucionais

O uso de códigos (ou fragmentos de códigos) convolucionais recursivos (convolucionais de resposta infinita) é importante para que se obtenha os melhores resultados de desempenho na codificação turbo.

Na versão mais simples, cada gerador de paridade pode ser considerado um código de taxa unitária. Assim, o gerador de paridade convolucional recursivo de memória m pode ser descrito por sua matrix geradora 1×1 com entrada do tipo $h(D)/g(D)$, onde $h(D)$ e $g(D)$ são polinômios de grau m no anel de polinômios sobre o campo algébrico binário $\{0, 1\}$. Sendo $u(D)$ e $c(D)$ os polinômios binários das sequências de entrada e saída do codificador, temos a relação entre os produtos na álgebra polinomial

$$g(D)\, c(D) = h(D)\, u(D). \tag{3.1}$$

Na prática, a implementação de um codificador recursivo é feita através da realimentação no sistema de registradores de deslocamento que geram a sequência de bits codificados $\mathbf{c}$ a

partir da sequência de bits de entrada **u**, como na figura 3.2 a seguir.

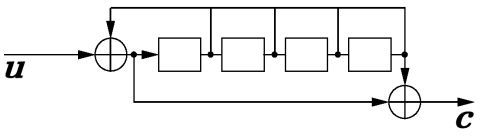

Figura 3.2: Gerador de paridade convolucional recursivo.

É opcional impor ou não um esquema de terminação no final da treliça de cada componente, para que o estado final seja conhecido na decodificação. Cada código componente de m memórias necessita de m transições controladas de estados para terminação, o que gera um conjunto de m bits de paridade a mais no final da sequência emitida. O envio de parte ou todos estes bits implica em alguma perda de taxa, desprezível para grandes comprimentos de bloco.

A opção mais simples é o truncamento da sequência de estados a cada final de bloco, reestabelecendo seu valor inicial a cada início de um novo bloco. A decodificação se realiza sem conhecimento prévio sobre o estados finais e não há perda de taxa, mas há menos de informação útil ao *backward* do BCJR. Para blocos grandes e bons entrelaçadores não há perda significativa de desempenho no truncamento. Em sua concepção original [9], questões de terminação não foram consideradas.

Os bits enviados para o canal são uma combinação dos bits de informação e os de paridades dos componentes. Vários esquemas de puncionamento, em que periodicamente bits codificados são desconsiderados e não enviados ao canal, podem ser empregados para se ajustar a taxa de codificação a um certo valor específico no projeto. A mais comum é a eliminação da metade de bits de paridade de cada componente, o que resulta numa taxa 1/2 para o sistema. O sistema turbo sem puncionamento tem taxa 1/3.

3.3.2. Entrelaçadores

Simulações mostram empiricamente que códigos turbo podem alcançar desempenhos próximos aos limites teóricos de probabilidade de erro em baixa relação sinal ruído, desde que o comprimento de bloco do código seja grande, e o entrelaçador seja pseudo-aleatório.

De fato, é o entrelaçador que define o comprimento de bloco num codificador turbo. O turbo original [9] considera um comprimento de bloco com $2^{16} = 65536$ bits de informação.

Entrelaçadores muito regulares, como o usual linha coluna, não resultam nos desempenhos excelentes dos obtidos por métodos pseudo-aleatórios. Entrelaçadores com bom espalhamento (entrelaçadores *S-random*) tem desempenhos ainda melhores.

Um entrelaçador *S-random* na formulação de Crozier [16] é um entrelaçador de bloco pseudo-aleatório tal que

$$|i - j| + |\pi(i) - \pi(j)| > S \tag{3.2}$$

para todo par i, j. A condição necessária para a existência é $S < \sqrt{2L}$, onde L é o comprimento de bloco. Em [16], Crozier apresenta um algoritmo rápido e elegante para obter entrelaçadores *S-random* deste tipo.

Adiante no capítulo, entraremos em maiores detalhes sobre o entrelaçamento.

3.3.3. O canal B-AWGN

Um canal B-AWGN é o resultado de uma sinalização binária num canal discreto no tempo e de amplitude contínua com ruído gaussiano. Em geral, considera-se sinalização com energia unitária $\{-1, +1\}$, e ruído gaussiano de variância σ^2. Desta forma, para uma variável binária C emitida para o canal, a saída será uma variável aleatória

$$Y = C + \sigma G, \tag{3.3}$$

onde G é uma variável gaussiana unitária.

A capacidade de um canal B-AWGN é

$$C = \frac{1}{2}\int_{-\infty}^{+\infty} p\left(y|-1\right)\log_2\frac{p\left(y|-1\right)}{p\left(y\right)}dy + \frac{1}{2}\int_{-\infty}^{+\infty} p\left(y|+1\right)\log_2\frac{p\left(y|+1\right)}{p\left(y\right)}dy \tag{3.4}$$

onde

$$p\left(y\right) = \frac{1}{2}p\left(y|-1\right) + \frac{1}{2}p\left(y|+1\right) \tag{3.5}$$

e

$$p\left(y|c\right) = \frac{1}{\sqrt{2\pi\sigma^2}}\exp\left\{-\frac{\left(y-c\right)^2}{2\sigma^2}\right\},\ c \in \{-1,+1\}. \tag{3.6}$$

Na aplicação dos limites da teoria de Shannon, num sistema codificado binário de taxa r, com probabilidade de erro de bit P_b, temos o limitante fundamental na forma

$$r\left(1-h\left(P_b\right)\right) < C \tag{3.7}$$

onde h é a função de entropia binária, $h\left(p\right) = -p\log_2 p - (1-p)\log_2(1-p)$.

A capacidade pode ser obtida e ficar como função de σ, ou de da razão $ebnodb = 10\log_{10} E_b/N_0$ em decibéis

$$\sigma = \frac{1}{\sqrt{2\ r\ 10^{ebnodb/10}}}. \tag{3.8}$$

3.4. Grafo-fator de um turbo padrão

Nesta seção, apresentamos uma descrição detalhada de um esquema básico de codificação turbo pela abordagem de grafos-fatores. A argumentação remete diretamente a idéias apresentadas no capítulo anterior.

A inserção do sistema padrão turbo no contexto de grafos-fatores e a decodificação turbo como caso particular do SP traz interpretações úteis e relevantes sobre o algoritmo turbo padrão clássico. A literatura no tema identifica a decodificação turbo como sendo um caso particular do SP [64] [53]. Mas uma interpretação mais detalhada sobre esta interseção e algumas nuances são pouco exploradas . O texto a seguir pretende contribuir neste sentido.

A figura 3.3 representa o grafo-fator de um bloco codificado num sistema de codificação turbo padrão. O entrelaçador e comprimento de bloco são muito pequenos ($L = 4$), longe dos

parâmetros práticos úteis. Mas o intuito do esquema é servir de referência para ilustrar as variáveis envolvidas e suas conexões. Quanto ao agrupamento dos nós similares, a disposição remete ao esquema da figura 3.1 de diagrama de blocos, com a inclusão de G e W, blocos referentes ao modelo de canal e a observações.

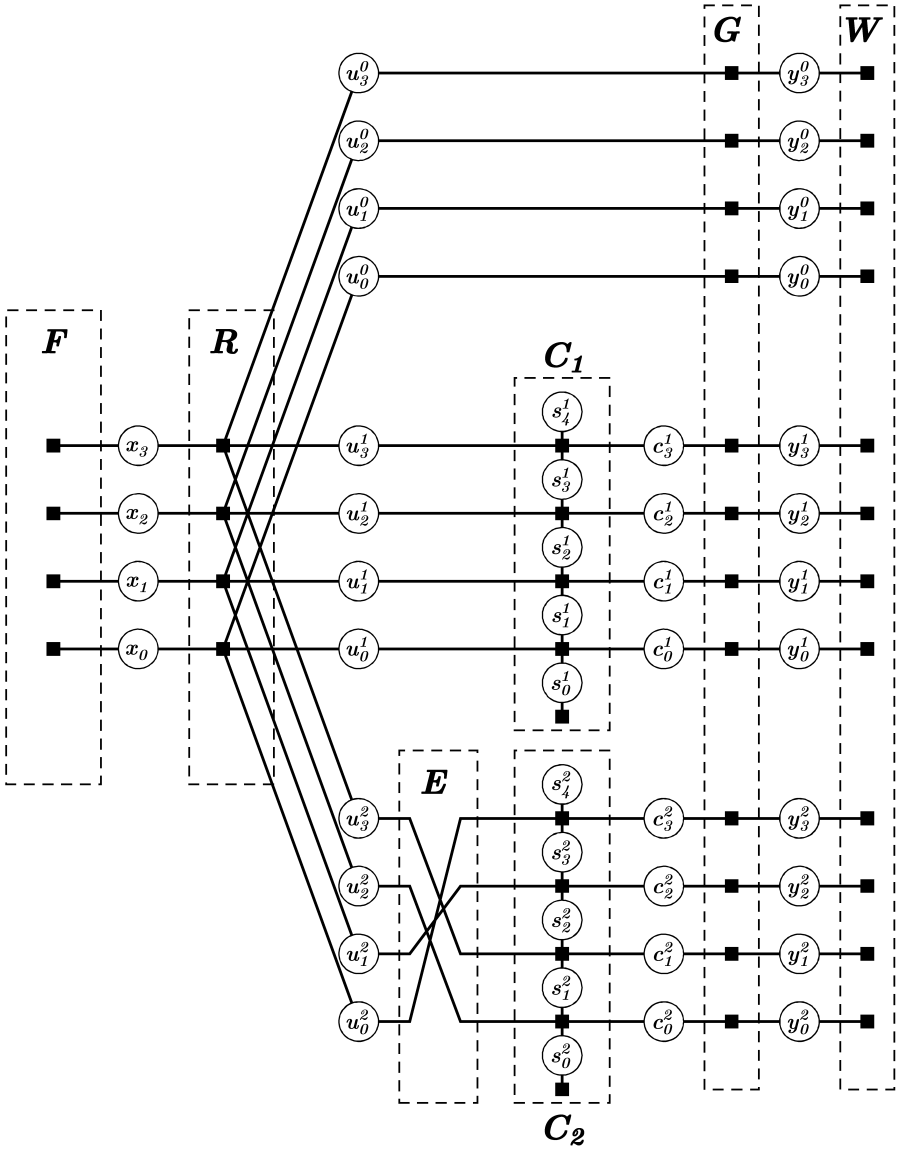

Figura 3.3: Grafo-fator de um sistema turbo.

Algumas características a se destacar neste esquema são enumeradas a seguir.

- As sequências de nós similares são dispostas verticalmente, com índices crescendo de

baixo para cima;

- As sequências de nós-função ficam internas a cada um dos blocos componentes;
- As sequências de nós-variáveis podem ser internas ou externas aos blocos componentes. As internas são relacionadas a estados da treliça dos codificadores. As externas são todas as outras variáveis, incluindo variáveis representando valores na saída do canal (variáveis com alfabeto $\mathbb{R}$).
- Usamos super-índices para indicar variáveis similares, nos três diferentes ramos da concatenação paralela, $k = 0, 1, 2$.

Como é interessante manter representada a distribuição conjunta do sistema completa para decodificação, com todas as suas variáveis e fatores, convenientemente incluímos nós referentes a características da fonte, do canal, e das observações de canal. Assim, temos a conjunta não só da parte de codificação, mas de todo o sistema de transmissão. A decodificação baseada em grafos-fatores resulta num decodificador teórico que considera os modelos de todos estes componentes.

Temos aqui uma realização generalizada por estados dos códigos componentes, e assim a evolução dos estados de suas treliças é dada pelas variáveis s_i^k, que remete diretamente à aplicação do SP em cada um deles, resultando no algoritmo *forward-backward* descrito em detalhes no capítulo anterior, que é essencialmente o mesmo BCJR da literatura turbo clássica [4].

O repetidor R, com seus nós-função triviais de igualdade, tem a função de misturar e distribuir adequadamente as diversas mensagens referentes a cada grupo de variáveis u_i^k de decodificação. Através de um SP degenerado (por se reduzir apenas a produtos de mensagens), ele administra a troca de mensagens entre os dois componentes de cálculos mais intensos C_1 e C_2, juntamente com a informação da parte sistemática do código das variáveis u_i^0.

O repetidor R é também o componente que ao final do processo iterativo fornece as mensagens para decisão que é feita nas variáveis x_i para terminar a decodificação.

O entrelaçador E não possui nós-função. Na representação por grafos-fatores, o entrelaçamento determina apenas diferentes linhas de conexões, determinadas de acordo com sua lei de permutação. Portanto, o conjunto entrelaçador/desentrelaçador na decodificação apenas realiza redirecionamentos pré-determinados de mensagens.

Na figura 3.4, é representado o mesmo esquema, mas de forma compacta para fins de descrição mais genérica de cada elemento, onde apenas as variáveis referentes a um dos índices sequenciais i estão presentes, e j é o respectivo índice associado pelo entrelaçador, ou seja $i = \pi(j)$, onde π é a lei de permutação do entrelaçador.

Com este tipo de representação, fica mais viável inserir também quais são as funções de cada nó-função, com seus significados probabilísticos bem estabelecidos. Isso é feito na figura 3.5.

É interessante manter a notação de distribuições de probabilidades nos nós-funções, uma vez que cada nó contém de fato as distribuições de probabilidades de cada grupo local de variáveis aos quais ele se conecta.

Analogamente ao que foi feito no capítulo anterior, esta realização pode ser considerada um tradução da rede bayesiana equivalente do sistema, com a ordem de causalidade entre as variáveis sendo mantida. Assim, mantém-se também uma fatoração da distribuição conjunta em distribuições condicionais e podemos representá-las como nós-funções.

Evidentemente, alguns dos nós-funções correspondem a relações determinísticas entre as suas variáveis. Mas ainda assim podem ser descritas como funções de probabilidades triviais, que assumem valores 0 ou 1. A vantagem é manter a descrição de tal forma que temos a fatoração da distribuição de probabilidade conjunta do sistema.

Os nós-funções dos códigos compontes e o repetidor são determinísticos. Mas ainda assim admitem esta representação, como probabilidades condicionais entre símbolos de entrada e símbolos de saída, assumindo valores 0 e 1. Temos assim uma visualização completa das funções que compõe a fatoração da distribuição conjunta.

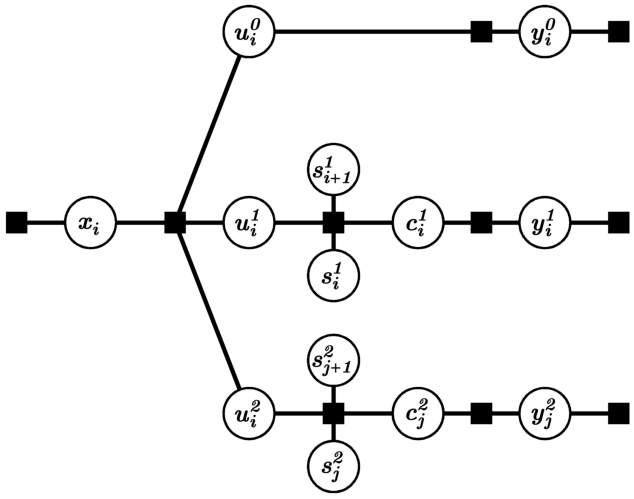

Figura 3.4: Fragmento do grafo-fator turbo.

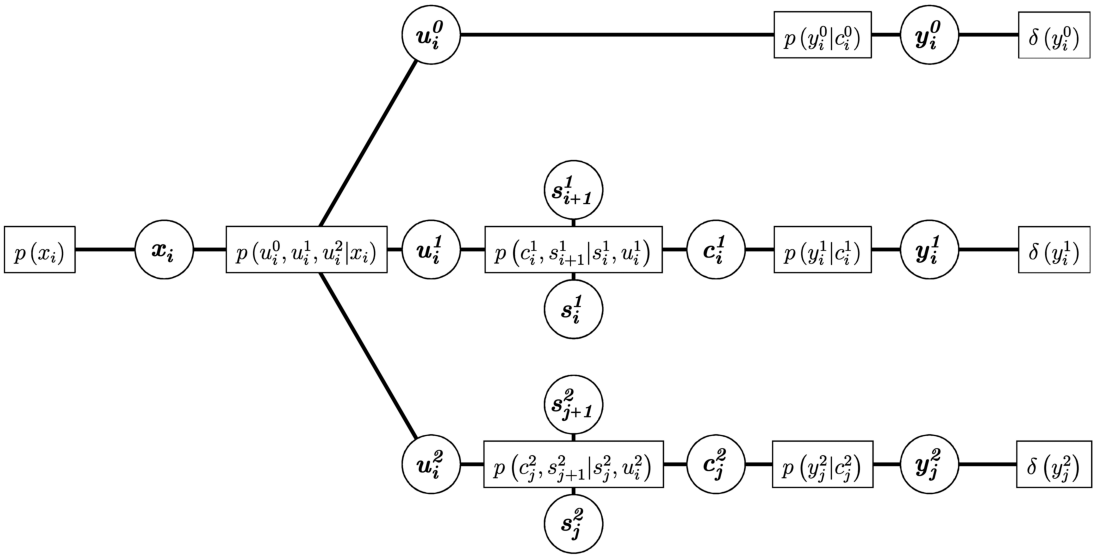

Figura 3.5: Fragmento do grafo-fator turbo e suas funções-fatores

A fonte tem os nós $p(x_i)$, usualmente probabilidades uniformes. Os nós do repetidor são nós de igualdade entre variáveis, mencionados no capítulo anterior, funções determinísticas simples, que no contexto probabilístico são fatores $p(u_i^0, u_i^1, u_i^2|x_i)$ para distribuição global do sistema.

Para códigos convolucionais, as funções $p\left(c_i^k, s_{i+1}^k|s_i^k, u_i^k\right)$ são funções indicadoras que

descrevem as associações na seção da treliça e ainda admitem fatoração adicional do tipo

$$p\left(c_i^k, s_{i+1}^k | s_i^k, u_i^k\right) = p\left(c_i^k | s_i^k, u_i^k\right) p\left(s_{i+1}^k | s_i^k, u_i^k\right), \tag{3.9}$$

onde $p\left(s_{i+1}^k | s_i^k, u_i^k\right)$ descreve a evolução dos estados, e $p\left(c_i^k | s_i^k, u_i^k\right)$ identifica o símbolo emitido em função do ramo da treliça. Todas estas representam relações determinísticas.

O modelo do canal fica implícito nas probabilidades de transição do tipo $p\left(y_i^k | c_i^k\right)$, $k = 0, 1, 2$. Aqui o índice k não indica mudanças de características das funções, já que todos os componentes k usam o mesmo canal.

Aqui, temos a inclusão dos nós-função que representam as observações do canal acopladas à direita no grafo, os nós $\delta\left(y_i^k\right)$. Em geral, estas são funções do tipo deltas de Dirac para observações de precisão infinita. Aqui admitimos esta recepção ideal. É interessante manter a idéia de um grafo-fator do qual a extração das marginais represente a decodificação. A inclusão dos nós $\delta\left(y_i^k\right)$ faz parte desta idéia. Além disso, esta montagem do grafo-fator abre a possibilidade de um modelo para o receptor que não seja este trivial. Um caso típico seria incluir o modelo da quantização na leitura dos valores observados no canal. Ou seja, este esquema para decodificação pelo grafo-fator pode incluir até o modelo do próprio receptor.

3.4.1. As mensagens na decodificação

Da forma como está construído o modelo de grafo-fator nesta seção para o sistema turbo, temos uma realização normal do grafo-fator. Na verdade, melhor do que isso, o esquema representado na seção anterior pode ser facilmente traduzido em um rede bayesiana equivalente com as mesmas variáveis. Uma realização com esta propriedade é sempre bem vinda, uma vez que leva a uma melhor interpretação sobre as relações de causalidade entre as variáveis envolvidas no grafo-fator, o que por sua vez resulta numa visão mais intuitiva de como se procede numa implementação prática a ordem de ocorrência das variáveis.

Portanto, neste tipo de realização, destaque para dois fatos relevantes:

- Temos uma ordem de causalidade muito bem definida, remetendo diretamente ao que foi discutido no capítulo anterior. Da forma como estão desenhados os grafos aqui, esta

ordem é da esquerda para direita para conexões horizontais, e de baixo para cima para conexões verticais. Com este direcionamento implícito, os ramos do grafo estabelecem portanto as relações de causalidade entre variáveis, uma propriedade similar aos ramos de rede bayesiana.

- Além disso, cada nó-variável tem no máximo duas conexões. O SP neste tipo de nó se simplifica, conforme já descrito no capítulo anterior.

Recapitulando a simplificação, considere um nó-variável com exatamente dois vizinhos, digamos $N(x) = \{f_1, f_2\}$. Neste caso, o nó x apenas repassa as mensagens entre seus nós vizinhos, devido às identidades

$$\mu_{f_2 \to x}(\cdot) \equiv \mu_{x \to f_1}(\cdot) \tag{3.10}$$

e

$$\mu_{f_1 \to x}(x) \equiv \mu_{x \to f_2}(x). \tag{3.11}$$

Ou seja, o nó x apenas repassa as mensagens entre seus nós vizinhos. É conveniente portanto identificar estas mensagens, reduzindo cada duas em uma só, com nova nomenclatura. São mensagens em ramos diferentes, mas que fluem no mesmo sentido do grafo, e são idênticas em conteúdo. Assim, podemos considerar que para cada variável há exatamente **duas** mensagens distintas associadas, que convenientemente denotaremos α e β, com características bem definidas de direcionamento, coerente com a direção sequencial de causalidade das variáveis. Numa analogia ao que foi apresentado no capítulo anterior, este tipo de esquematização do sistema turbo permite a classificação de todas as mensagens que fluem através das conexões em relação a duas direções bem definidas. Mensagens tipo α fluem na direção da causalidade (*forward*), e mensagens β fluem na direção oposta (*backward*). Portanto, em cada nó-variável, temos a caracterização das mensagens que passam através dela (no máximo duas, uma em cada sentido). Uma é do tipo α e outra é do tipo β.

A figura 3.6 é uma nova versão das anteriores, agora dando ênfase à identificação e caracterização das mensagens associadas aos nós-variáveis e miniaturizando a representação dos nós do grafo. São mostradas as mensagens mais relevantes para um decodificação usual.

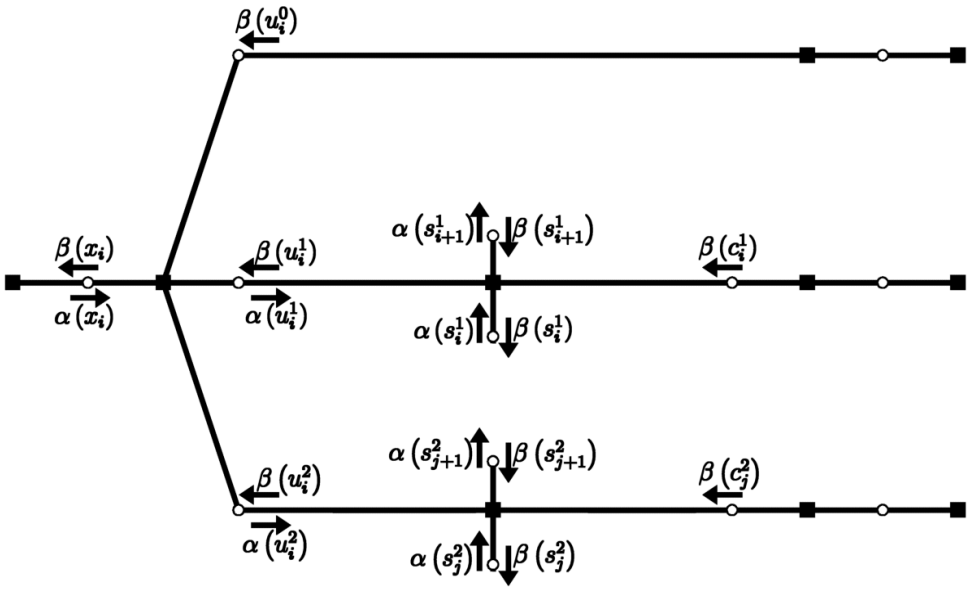

Figura 3.6: Fragmento do grafo-fator turbo, com ênfase nas mensagens.

Algumas observações e caracterizações das mensagens e seus papéis:

- Mensagens $\alpha(x_i)$ e $\beta(x_i)$: são mensagens exteriores aos ciclos do grafo e portanto não são recalculadas no processo iterativo. As mensagens $\alpha(x_i)$ são simplesmente as probabilidades *a priori* da fonte. As mensagens $\beta(x_i)$ são calculadas somente ao término das iterações e o produto $\alpha(x_i)\beta(x_i)$ é a função para decisão, isto é,

$$\hat{x}_i = \underset{x_i}{\operatorname{argmax}}\, \alpha(x_i)\beta(x_i) \ . \tag{3.12}$$

- Mensagens $\beta(u_i^0)$, $\beta(c_i^1)$, e $\beta(c_i^2)$: também são mensagens exteriores aos ciclos do grafo (canal sem memória) e são calculadas apenas no início, a partir do modelo do canal e das observações.

- $\alpha\left(u_i^k\right)$ e $\beta\left(u_i^k\right)$: são todas participantes dos ciclos do grafo e portanto seguem sendo recalculadas a cada iteração. $\beta\left(u_i^k\right)$ são as chamadas informações extrínsecas. No grafo com ciclos, não há mais interpretação probabilística precisa sobre as mensagens, mas ainda assim, podemos considerar a aproximação

$$\beta\left(u_i^k\right) \simeq p\left(y_0^k, \ldots, y_L^k | u_i^k\right), \ k = 1, 2 \tag{3.13}$$

e que as extrínsecas representam aproximadamente parametrizações de funções de probabilidades condicionais deste tipo. A interpretação probabilística das mensagens do BCJR dada no capítulo anterior agora não é exatamente válida com o acoplamento do sistema e iteração entre blocos, mas ainda assim é valiosa se pensada como uma boa aproximação das características das mensagens que fluem na decodificação turbo.

- $\alpha\left(s_i^k\right)$ e $\beta\left(s_i^k\right)$: quase todas participantes dos ciclos do grafo. O *forward-backward* é o cronograma interno a cada um dos códigos componentes para o cálculo eficiente destas mensagens e consiste no cálculo sequencial destas mensagens, $\alpha\left(s_i^k\right)$ com i crescente e $\beta\left(s_i^k\right)$ com i decrescente.

Se L é o comprimento de bloco, a decodificação é feita no intervalo $i = 0, \ldots, L-1$. O processo *forward* para cada decodificador consiste no cálculo, a partir da mensagem $\alpha\left(s_0^k\right)$, das consecutivas $\alpha\left(s_1^k\right), \ldots, \alpha\left(s_{L-1}^k\right)$. O processo *backward* para cada decodificador consiste no cálculo, a partir da mensagem $\beta\left(s_L^k\right)$, das consecutivas $\beta\left(s_{L-1}^k\right), \ldots, \beta\left(s_1^k\right)$.

Assim, as mensagens $\alpha\left(s_0^k\right)$ e $\beta\left(s_L^k\right)$ pertencem a ramos fora da parte cíclica. São dadas *a priori*, com informações sobre início e términos das treliças. A mensagen $\alpha\left(s_0^k\right)$ carrega a certeza de início da treliça num estado conhecido. $\beta\left(s_L^k\right)$ é identicamente unitária para o caso de treliças truncadas. O caso de treliças terminadas implicaria numa inserção de variáveis adicionais na parte final das sequências s_i^k no grafo e a perda de uma certa regularidade no grafo.

É usual desenhar o grafo do sistema omitindo-se os nós-função que representam as observações do canal. De fato, na literatura, nem mesmo o canal com suas variáveis Y_i são inseridas no grafo. Aqui, a opção por representá-los como parte integrante do grafo vem da intenção de manter a idéia geral de que estamos fatorando uma função global conjunta. Como vimos no capítulo anterior, quando se fala em marginalização da conjunta para decodificação, esta conjunta implicitamente deve incluir estes fatores.

De fato, para fins de implementação prática, é como se o SP nestas ramificações finais do grafo fosse "pré-processado", já que Y_i são variáveis contínuas. Daí vem a confusão. Ao mesmo tempo que Y_i são variáveis contínuas, o fato de termos deltas $\delta\left(y - y_{observ}\right)$ como

modelo de observação simplifica tudo quando teoricamente os cálculos passam pelos nós $p(y|c)$, e a mensagem $\beta(c)$ chega. Ou seja, para estes ramos na extremidade direita do grafo, o algoritmo SP (que se torna integral-produto pela natureza contínua da variável) fornece

$$\begin{aligned} \beta(c) &= \int p(y|c)\,\beta(y)\,dy \\ &= \int p(y|c)\,\delta(y - y_{observ})\,dy \end{aligned} \tag{3.14}$$

o que resulta em $\beta(c) = p(y_{observ}|c)$, ou seja, a mensagem $\beta(\cdot)$ coincide ponto a ponto no alfabeto com $p(y_{observ}|\cdot)$, a função do nó que modela o canal, calculada com y_{observ} como parâmetro fixo. Portanto, quando se fala em assumir y_{observ} como parâmetro fixo na decodificação, o conceito real empregado é na verdade o "pré-processamento" do SP nesta parte do grafo, dado pela integral acima. Mas ainda assim, na essência, permanecemos lidando com uma marginalização global.

Um caso particular e mais degenerado ainda é o caso de variáveis com apenas uma conexão. São variáveis isoladas, que enviam mensagem unitária para o resto do grafo e portanto em nada influenciam qualquer marginalização. Ainda assim, esta pode ser considerada uma mensagem do tipo α ou do tipo β. No grafo de decodificação do turbo padrão concebido aqui, as variáveis de apenas uma conexão são as que representam os estados finais das treliças (não terminadas) S_L^1 e S_L^2. Numa versão do grafo para considerar puncionamento de bits codificados, as variáveis C_i^k puncionadas seriam também variáveis deste tipo. Nestes dois exemplos, $\beta = 1$ é enviada para o resto do grafo pela variável. Usualmente, estas variáveis não costumam ser inferidas na decodificação. Portanto suas mensagens α não são usadas (nem precisam ser calculadas, embora existam e sejam bem definidas).

3.4.2. Cronogramas

Para fins de referência e para fixar notação, um resumo do cronograma (*scheduling*) baseado na figura 3.6 equivalente à decodificação turbo clássica é sumarizado na tabela 3.1 a seguir. Entre parênteses, a correspondente operação na decodificação clássica. L é o comprimento do bloco.

(1) obtenção de $\beta\left(u_i^0\right)$, $\beta\left(c_i^1\right)$, e $\beta\left(c_i^2\right)$, $i = 0, \ldots, L-1$;
(2) atualiza $\alpha\left(u_i^1\right)$, $i = 0, \ldots, L-1$ (*a priori* para C_1);
(3) atualiza $\alpha\left(s_i^1\right)$ e $\beta\left(s_i^1\right)$, $i = 0, \ldots, L-1$ (*forward-backward* dentro de C_1);
(4) atualiza $\beta\left(u_i^1\right)$, $i = 0, \ldots, L-1$ (extrínsecas de C_1);
(5) atualiza $\alpha\left(u_i^2\right)$, $i = 0, \ldots, L-1$ (*a priori* para C_2);
(6) atualiza $\alpha\left(s_j^2\right)$ e $\beta\left(s_j^2\right)$, $j = 0, \ldots, L-1$ (*forward-backward* dentro de C_2);
(7) atualiza $\beta\left(u_i^2\right)$, $i = 0, \ldots, L-1$ (extrínsecas de C_2);
(8) se critério de parada não é satisfeito, volta para (2);
(9) obtenção de $\beta\left(x_i\right)$, $i = 0, \ldots, L-1$;
(10) decisão $\operatorname{argmax} \alpha\left(x_i\right) \beta\left(x_i\right)$, $i = 0, \ldots, L-1$.

Tabela 3.1: Cronograma clássico

Neste sumário, deve ficar subentendido que as mensagens β nas etapas (3) e (6) devem ser calculadas na ordem reversa do índices.

Portanto, temos explicitado o cronograma geral do turbo clássico em função de mensagens características do tipo α e β em todas as variáveis e uma visão geral sobre os dois fluxos de mensagens direcionados bem definidos ao longo de todo o grafo.

O fato de que as mensagens variam ao longo das iterações sugere que eventualmente seria conveniente para uma descrição mais detalhada um indexador adicional para denotar uma dada mensagem em uma dada iteração. Entretanto, podemos evitar sobrecarregamento na notação desde que fique subentendido que as mensagens são dinâmicas e variam a cada iteração.

Também útil para futuras referências, a seguir um resumo do cronograma de decodificação paralela na tabela 3.2, que realiza cálculos de mensagens simetricamente em relação a C_1 e C_2, tem as mesmas etapas do cronograma padrão, mas em ordem distinta.

Aqui, temos os pares de tarefas independentes e similares, uma em cada componente. Ou seja: (2) e (3) são independentes; (4) e (5) são independentes; (6) e (7) são independentes.

(1) obtenção de $\beta\left(u_i^0\right)$, $\beta\left(c_i^1\right)$, e $\beta\left(c_i^2\right)$, $i = 0, \ldots, L-1$;
(2) atualiza $\alpha\left(u_i^1\right)$, $i = 0, \ldots, L-1$ (*a priori* para C_1);
(3) atualiza $\alpha\left(u_i^2\right)$, $i = 0, \ldots, L-1$ (*a priori* para C_2);
(4) atualiza $\alpha\left(s_i^1\right)$ e $\beta\left(s_i^1\right)$, $i = 0, \ldots, L-1$ (*forward-backward* dentro de C_1);
(5) atualiza $\alpha\left(s_j^2\right)$ e $\beta\left(s_j^2\right)$, $j = 0, \ldots, L-1$ (*forward-backward* dentro de C_2);
(6) atualiza $\beta\left(u_i^1\right)$, $i = 0, \ldots, L-1$ (extrínsecas de C_1);
(7) atualiza $\beta\left(u_i^2\right)$, $i = 0, \ldots, L-1$ (extrínsecas de C_2);
(8) se critério de parada não é satisfeito, volta para (2);
(9) obtenção de $\beta\left(x_i\right)$, $i = 0, \ldots, L-1$;
(10) decisão $\operatorname{argmax} \alpha\left(x_i\right) \beta\left(x_i\right)$, $i = 0, \ldots, L-1$.

Tabela 3.2: Cronograma paralelo

3.4.3. Simulação do turbo padrão com o algortimo SP

A figura 3.7 ilustra o desempenho obtido por uma decodificação baseada no grafo-fator, algoritmo SP e cronograma clássico dado na seção anterior, quando aplicados num turbo com parâmetros similares aos parâmetros usados em [9]. Ou seja, mesmo comprimento de bloco 65536 bits, mesmos geradores de paridade com treliça de 16 estados, com razão de polinômios $(1 + D^4) / (1 + D + D^2 + D^3 + D^4)$, e uma taxa global 1/2 por puncionamento. Estão apresentadas na figura as curvas de desempenho para diferentes números de iterações antes das decisões: 3, 6, 9, 12, 15 e 18 iterações no algoritmo. Como usual, são curvas de taxa de erro de bits de informação (*bit error rate*, BER) em escala logarítmica vs. relação sinal-ruído por bit de informação, denotada por SNR (o mesmo que E_b/N_0).

A diferença significativa é o tipo de entrelaçamento empregado. Aqui foram usados entrelaçadores *S-random* com espalhamentos mínimos de 128 bits, obtidos pelo método de Crozier [16]. A cada bloco simulado, um novo entrelaçador foi sorteado.

Para a curva de 18 iterações, pontos no final da região de *water fall* estão indicados. Como é de se esperar, o resultado de uma decodificação baseada no algoritmo SP sobre o grafo-fator fornece resultados de desempenho parecidos aos obtidos pelo algoritmo de decodificação clássico (alguns ligeiramente melhores, devido à escolha de entrelaçadores *S-random*). Para

os pontos de maior interesse no gráfico, mais de 10^9 bits de informação foram simulados para obtenção da frequência de erros. Como exemplo, para o ponto 0,66dB, simulamos 2421 blocos, 158662656 bits de informação, e obtivemos 605 decisões de bits erradas no algoritmo para 18 iterações fixas.

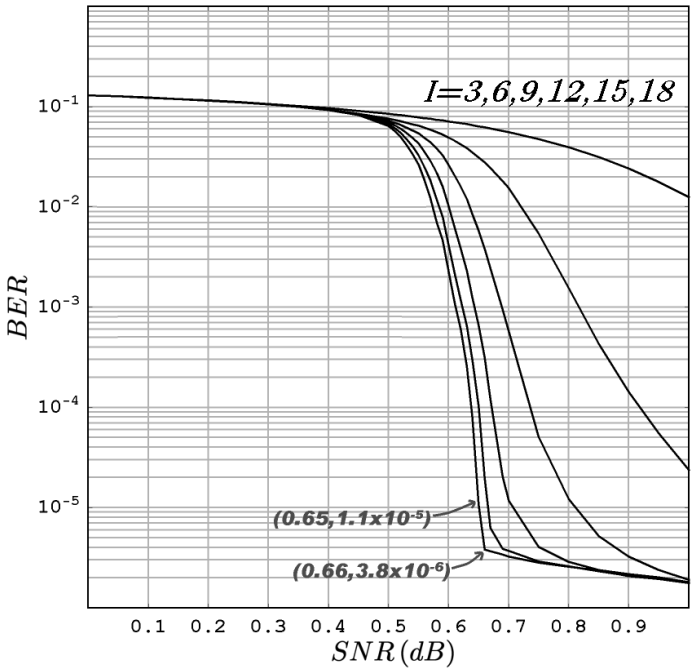

Figura 3.7: Simulação de desempenho do SP no código turbo. Curvas para diferentes número de iterações, $I = 3, 6, 9, 12, 15, 18$.

3.5. Codificação e decodificação turbo contínua: sistemas *stream-oriented*

Entre as inúmeras variações de sistemas envolvendo o princípio turbo, está o caso da codificação e decodificação turbo que não é realizada em blocos, e sim de forma contínua. Uma explanação completa pode ser encontrada na principal linha de textos sobre o tema, a maioria com a nomenclatura *stream-oriented* [42] , [43]. Aqui, vamos descrever brevemente como se estabelece este paradigma "turbo contínuo".

Códigos turbo foram originalmente projetados usando grande comprimento de bloco para evidenciar seu poder de aproximação dos limites teóricos absolutos de Shannon. Posteriormente, foi mostrado que códigos turbo e similares bem projetados tem desempenhos próximos aos limites teóricos para uma ampla faixa de valores de parâmetros de projetos de código, como a taxa e o comprimento de bloco, conforme foi estabelecido em [20]. Códigos turbo são subótimos mesmo quando os comprimentos são modestos.

3.5.1. Comprimentos de bloco e atraso de transmissão

Na teoria clássica de códigos turbo, a principal preocupação não é a implementação prática em sistemas reais. Não são considerados questões de atraso de codificação, causalidade no entrelaçamento, ou administração de sincronismo em geral.

A teoria turbo originalmente considera grandes blocos, que quando codificados e completamente transmitidos em toda sua extensão, seriam decodificados com intensos cálculos. Não há preocupação com o uso eficiente de um canal real, onde em alguns casos é desejável que dados sendo continuamente gerados devem ser transmitidos na mesma velocidade, sem grandes oscilações entre uso intenso e ociosidade do canal.

Podemos considerar que, em geral, há aplicações práticas nas quais não é conveniente administrar grandes blocos de dados, seja na codificação, na transmissão ou na decodificação.

Uma questão prática importante envolvendo grandes blocos é o atraso. Grande parte das aplicações práticas demandam mínimo atraso de transmissão e códigos com grande comprimento de bloco são incovenientes.

Em situações nas quais a principal preocupação é o atraso, não cabe mais pensar numa codificação com grandes blocos e proximidade dos limites absolutos de Shannon. Podemos pensar apenas em limites relativos, que consideram o comprimento de bloco como parâmetro.

Ainda assim, códigos concatenados com decodificação iterativa fornecem bons desempenhos, relativamente ao melhor que se pode fazer dadas as restrições impostas.

Originalmente, códigos turbo foram concebidos como códigos de bloco, de grande

comprimento e entrelaçadores com alta irregularidade. Pode-se dizer que o entrelaçador é o elemento chave de um esquema turbo tradicional. Ele é o elemento que de fato determina o comprimento de bloco.

Neste contexto de bloco pequeno, a irregularidade do entrelaçador já não faz tanta diferença no desempenho. Pode ser mais interessante o uso de entrelaçadores com regras determinísticas bens estabelecidas e com boas propriedades de atraso.

3.5.2. Codificação contínua

Vamos estabelecer então a diferenciação entre codificação turbo contínua e a tradicional codificação turbo em blocos.

O termo codificação turbo contínua é relativo à evolução livre das treliças convolucionais dos codificadores. A essência da codificação turbo em blocos é a segmentação em blocos.

Na codificação turbo padrão, os dados são particionados em segmentos disjuntos, sem superposição entre eles. Cada um destes segmentos é manipulado de ponta a ponta independentemente, desde a codificação até a decodificação. Ou seja, a codificação turbo em blocos equivale a um código de bloco, e a decodificação também é realizada de forma compatível. Assim, tanto a codificação quanto a decodificação são realizadas internamente a cada segmento de bits.

Na codificação turbo em blocos, o mais conveniente é reiniciar os estados dos codificadores convolucionais a cada bloco, zerando-os. Eventualmente, o desempenho do sistema pode ser sensível à terminação das treliças em estados também conhecidos para a decodificação, sobretudo em modestos comprimentos de blocos. Isso implica num incremento de detalhes estruturais para efetuar a terminação num caso de implementação prática, sobretudo pela natureza convolucional dos códigos componentes típicos do turbo. Também busca-se projetar o entrelaçador para evitar que efeitos de bordo resultem em perda de desempenho.

Na verdade, como todo o foco central de um sistema turbo tradicional é sua decodificação especial (feita em blocos), a codificação é pensada no sentido de melhorar o desempenho

final desta decodificação. Assim, são induzidos início e término nas treliças convolucionais e entrelaçadores irregulares são usados.

Em contraste à codificação por blocos, a codificação contínua retira todos estes dilemas. Não há divisões ou bordas, presentes na codificação turbo clássica.

A característica convolucional dos codificadores é plena, pois evoluem livres e continuamente. Portanto, no turbo contínuo, as treliças convolucionais evoluem livremente com suas seções invariantes.

Como consequência imediata, desfaz-se a obrigatoriedade do entrelaçador de bloco. Qualquer entrelaçador periódico serve para codificação contínua.

3.5.3. Entrelaçamento "contínuo"

Entrelaçadores convolucionais são os entrelaçadores "não-bloco" mais simples.

A maior motivação para o desenvolvimento da codificação turbo contínua vem da idéia de poder usar entrelaçadores convolucionais com a finalidade de aproveitar suas propriedades práticas. Além da facilidade de implementação, o bom gerenciamento sob necessidade de sincronismo e baixa latência refletem numa operação cadenciada de geração de dados para transmissão.

Em última instância, num sistema turbo, o atraso essencial de codificação/decodificação é determinado pelo esquema de entrelaçamento, o que numa aplicação prática é relevante, e em alguns casos, até mais importante do que a otimização de outras variáveis.

Entrelaçadores convolucionais oferecem redução da latência de transmissão através de entrelaçamento de baixo período e baixo atraso essencial de entrelaçamento/desentrelaçamento.

Fora do ambiente turbo, esquemas de entrelaçamento mais comuns na prática já utilizam entrelaçadores convolucionais, pela sua simplicidade de implementação, descrição e propriedades conhecidas de espalhamento, bom sincronismo e baixa latência.

Já no contexto da codificação turbo, os entrelaçadores são parte integrante do código, determinando várias características dele.

Uma vez que o turbo original é uma concatenação de códigos convolucionais, é natural a idéia de conservar e estender a característica convolucional para todo o conjunto, determinando uma codificação convolucional globalmente. Mais uma consequência interessante do uso de entrelaçadores convolucionais.

De forma geral, o paradigma *stream-oriented* consiste na junção de codificação contínua e entrelaçadores periódicos arbitrários. Pode ser pensado como uma forma de generalização para codificação turbo visando sua implementação em sistemas reais, na medida em que oferece uma visão geral de como lidar com qualquer esquema regular de entrelaçamento que se queira forjar. A única exigência é que seja periódico, o mínimo que se espera de qualquer esquema de entrelaçamento.

Um detalhe a se destacar é que a codificação contínua não exclui o uso de entrelaçadores de bloco. Entrelaçadores de bloco não implicam necessariamente numa codificação por blocos, embora seja recomendável para melhor eficiência. Além disso, o paradigma *stream-oriented* é útil para focalizar melhor alguns conceitos que ficam em segundo plano no turbo padrão, tais como causalidade na codificação e atraso real na codificação e decodificação.

Essa abordagem nos leva também a considerar que a indexação temporal dos símbolos é agora contínua, ou seja, sequências infinitas ou semi-infinitas, embora sejam manipuladas finitamente nos seus períodos e múltiplos.

3.5.4. Arquitetura de decodificação

A natureza contínua da codificação induz uma decodificação diferenciada. Pelo caráter sequencial e regular da redundância que um código convolucional gera, a decodificação eficiente por algoritmos do tipo *forward-backward* ainda é possível e recomendada, mas devem ser adaptados para a situação. E isso significa usar versões janelas deslizantes destes algoritmos.

Na literatura existente no tema, cuja principal referência é [43], a decodificação é baseada feita por blocos SISO, em um esquema sequencial de blocos de decodificação, uma arquitetura por "tubulações" conectadas (*pipeline*), onde cada componente realiza uma iteração a partir da informação que recebe do anterior, e passa nova informação adiante para o próximo. E cada um destes componentes decodificadores usa uma versão "janela deslizante" do algoritmo BCJR MAP.

Assim, a arquitetura *pipeline* consiste numa abstrata clonagem múltipla de um módulo decodificador padrão do turbo (os dois decodificadores das treliças, entrelaçador e desentrelaçador), e a concatenação serial destes módulos, para cada um realizar cálculos referentes a uma iteração e passar resultados adiante para o próximo módulo. Assim, temos módulos idênticos, mas cada um com seu estágio particular de estimação das variáveis, cada um realizando uma instância independente do algoritmo BCJR.

Portanto, podemos dizer que esta arquitetura de decodificação *stream* esquematizada na literatura é um tanto complicada para implementação prática, pois demanda módulos independentes múltiplos e vários *buffers* de memória entre eles.

3.6. Teoria geral de entrelaçadores

Nesta seção, apresentamos os principais fundamentos da teoria de entrelaçadores, com ênfase nos conceitos mais utilizados na sequência do texto. Uma exposição mais detalhada do tema pode ser encontrada em [45], ou nos textos clássicos [72] e [28].

Um entrelaçamento de uma sequência de símbolos $(x_i)_{i\in\mathbb{Z}}$ consiste numa reordenação da sequência de acordo com uma dada função de permutação. Um entrelaçador é um dispositivo num sistema de comunicação que a partir de sequência de símbolos como entrada devolve como saída o entrelaçamento da sequência.

Formalmente, um entrelaçador opera sobre sequências de símbolos. Entretanto, é comum descrevê-lo por sua ação sobre a indexação da sequência.

A descrição de entrelaçamento também independe do alfabeto da sequência. Para todos

os efeitos, seja $\mathfrak{A}$ este alfabeto. Logo, uma representação de uma sequência é qualquer mapa $x : \mathbb{Z} \to \mathfrak{A}$. O conjunto destes mapas é denotado $\mathfrak{A}^{\mathbb{Z}}$. Dois mapas $x_1, x_2 \in \mathfrak{A}^{\mathbb{Z}}$ representam a mesma sequência se são o mesmo mapa a menos de translação, isto é, para alguma constante $c \in \mathbb{Z}$, $x_1(i) = x_2(i + c)$ para qualquer i. No que segue, $(x_i)_{i\in\mathbb{Z}}$ e similares são representações de sequências.

Uma representação de um entrelaçador pode ser dada por uma função de permutação, $\pi : \mathbb{Z} \to \mathbb{Z}$, tal que sequências de entrada $(x_i)_{i\in\mathbb{Z}}$ e sequências de saída $(y_i)_{i\in\mathbb{Z}}$ se relacionam por

$$y_i = x_{\pi(i)}. \tag{3.15}$$

Evidentemente, π é uma função biunívoca.

Um entrelaçador, visto como um dispositivo síncrono, no instante i lê o símbolo de entrada x_i e emite um símbolo $y_i = x_{\pi(i)}$.

Formalmente, a representação por funções de permutação não é única. Se π representa um entrelaçamento, qualquer translação de π representa o mesmo entrelaçamento. Precisamente, $\pi(i + c_1) + c_2$ representa o mesmo entrelaçamento que $\pi(i)$, para quaisquer constantes c_1 e c_2, pois representam o mesmo tipo de mapa entre sequências.

A cada entrelaçador, o desentrelaçador associado é o mapa inverso entre sequências. Duas funções de permutação π_1 e π_2 representam entrelaçador e desentrelaçador, se e somente se

$$\pi_2(\pi_1(i)) = i + c \tag{3.16}$$

para alguma constante $c \in \mathbb{Z}$. Ou seja, a composição $\pi_2 \circ \pi_1$ é uma translação.

Com as definições gerais acima, podemos definir algumas características das funções π, e seus reflexos no entrelaçamento que elas representam.

Por definição, um entrelaçador é **periódico** quando existe um inteiro positivo N tal que

$$\pi(i) \bmod N = \pi(i + N) \bmod N, \ \forall i \in \mathbb{Z}, \tag{3.17}$$

e seu período é o menor N que satisfaz a propriedade acima. Equivalentemente, dizemos que

um entrelaçador é periódico e com período N, quando sua função de permutação satisfaz

$$\pi(i+N) = \pi(i) + N, \ \forall i \in \mathbb{Z}. \tag{3.18}$$

Na prática, a periodicidade é uma propriedade mínima exigida para entrelaçadores. São entrelaçadores tais que seus mapas $\pi : \mathbb{Z} \to \mathbb{Z}$ tem complexidade finita.

O desentrelaçador de um entrelaçador periódico é também periódico, e com mesmo período.

Uma função de permutação π é dita **causal** quando

$$\pi(i) \leq i, \ \forall i \in \mathbb{Z}. \tag{3.19}$$

Para fins práticos, é sempre desejável um entrelaçamento com função de permutação causal. Essa formulação matemática do princípio de causalidade traduz que os símbolos na saída do entrelaçador só existem depois que o associado símbolo na entrada ocorre. Dada uma $\pi(i)$, é sempre possível obter uma equivalente $\pi(i+c_1)+c_2$ causal, desde que haja a hipótese mínima de periodicidade.

Um conceito de interesse em um entrelaçamento é o atraso que ele gera. Por definição, a **função de atraso** de uma permutação é dada ponto a ponto pela relação

$$d(i) = i - \pi(i). \tag{3.20}$$

O mapa d descreve ponto a ponto o atraso de cada símbolo na saída do entrelaçador, pois temos a identidade

$$y_i = x_{i-d(i)} \ . \tag{3.21}$$

Para qualquer entrelaçador periódico de período N, sua função de atraso satisfaz

$$\begin{aligned} d(i+N) &= (i+N) - \pi(i+N) \\ &= (i+N) - (\pi(i)+N) \\ &= i - \pi(i) \\ &= d(i). \end{aligned} \tag{3.22}$$

Para uma função de permutação π causal, $d(i) \geq 0$, $\forall i \in \mathbb{Z}$.

Por definição, a **latência** de um entrelaçador é dada por

$$\begin{aligned} D &= \sup_i d(i) - \inf_i d(i) \\ &= \sup_i \{i - \pi(i)\} - \inf_i \{i - \pi(i)\} \ . \end{aligned} \tag{3.23}$$

A latência do respectivo desentrelaçador coincide em valor com a latência do entrelaçador.

A noção de periodicidade e latência são intrísecos ao entrelaçador, embora sejam usualmente definidas a partir de uma função de permutação que o representa. Já a definição de causalidade é inerente a funções de permutação e não ao entrelaçador que ela realiza.

Por definição, uma função de permutação é dita **canônica** quando é causal e seu atraso global é mínimo, isto é,

$$\inf_i \{i - \pi(i)\} = 0 \ . \tag{3.24}$$

Dada uma $\pi(i)$, é sempre possível obter uma equivalente $\pi(i + c_1) + c_2$ canônica, desde que haja a hipótese mínima de periodicidade. Portanto, qualquer entrelaçador periódico tem uma realização canônica e seu respectivo desentrelaçador canônico.

Temos o seguinte resultado fundamental: Sejam π e π^* realizações canônicas de um entrelaçador periódico e seu respectivo desentrelaçador. Então,

$$\pi^*(\pi(i)) = i - D. \tag{3.25}$$

Ou seja, realizações canônicas são tais que o mapa composto tem o mínimo atraso, e este atraso é D, a latência do par entrelaçador/desentrelaçador.

3.6.1. Representação geral de entrelaçadores periódicos

A função de permutação de qualquer entrelaçador periódico de período N pode ser decomposta e representada por

$$\pi(i) = \varphi(i \bmod N) - N \times \psi(i \bmod N) + \left\lfloor \frac{i}{N} \right\rfloor N, \tag{3.26}$$

onde $\varphi : \{0, \ldots, N-1\} \to \{0, \ldots, N-1\}$ e $\psi : \{0, \ldots, N-1\} \to \mathbb{Z}^+$, com φ bijetora e ψ injetora.

A decomposição fundamental acima permite diferenciar entre entrelaçadores de bloco e não-bloco pela caracterização das três partes desta expressão.

Entrelaçadores de bloco são tais que $\psi \equiv 0$. Caso contrário, temos um entrelaçador não-bloco.

3.6.2. Entrelaçadores convolucionais

Entrelaçadores convolucionais são aqueles em que, além de $\psi \neq 0$, α é a identidade. Assim, a primeira parcela e a última parcela na expressão podem ser agrupadas,

$$i \bmod N + \left\lfloor \frac{i}{N} \right\rfloor N = i \tag{3.27}$$

e o entrelaçador convolucional tem a lei geral de formação

$$\pi(i) = i - N \times \psi(i \bmod N). \tag{3.28}$$

Essa expressão é a formulação generalizada de entrelaçadores convolucionais, com a sigla GCI (*Generalized Convolutional Interleavers*) atribuída em [43].

Cada função ψ define um esquema de entrelaçamento convolucional generalizado. As restrições sobre ela são apenas que não tenha valores negativos, para termos causalidade. Evidentemente, a composição com o argumento $i \bmod N$ implica que apenas seus valores $\psi(0), \ldots, \psi(N-1)$ são efetivos para definir o entrelaçamento.

Para uma realização canônica, devemos ter

$$\text{mín}\{\psi(0), \ldots, \psi(N-1)\} = 0. \tag{3.29}$$

Seja

$$\psi_{\text{máx}} = \text{máx}\{\psi(0), \ldots, \psi(N-1)\} \tag{3.30}$$

Temos a relação fundamental para o entrelaçamento convolucional generalizado

$$D = N\psi_{\text{máx}}. \tag{3.31}$$

O desentrelaçador canônico é dado por

$$\pi(i) = i - N \times (\psi_{\text{máx}} - \psi(i \bmod N)). \tag{3.32}$$

O entrelaçador convolucional mais simples é um caso particular da expressão (3.28), com $\beta(\cdot)$ uma função linear de coeficiente B. Assim, a função de permutação é dada por

$$\pi_{N,B}(i) = i - N \times B \times (i \bmod N), \tag{3.33}$$

onde B é um parâmetro constante característico. Os parâmetros constantes N e B caracterizam unicamente a lei $\pi_{N,B}(i)$ do entrelaçador. O parâmetro B pode ser denominado multiplicidade, pois a concatenação em série de B entrelaçadores idênticos do tipo $\pi_{N,1}(i)$ resulta num entrelaçador global $\pi_{N,B}(i)$.

Pode-se dizer que este tipo de entrelaçador convolucional é o equivalente não-bloco do entrelaçador linha-coluna, pelas suas características de espalhamento semelhantes.

A função de permutação (3.33) é uma realização canônica. A respectiva realização canônica do desentrelaçador é

$$\begin{aligned} \pi^*_{N,B}(i) &= i - N \times B \times ((N-1) - i \bmod N) \\ &= i + N \times B \times (i \bmod N) - N \times B \times (N-1), \end{aligned} \tag{3.34}$$

e a latência do par é

$$D = N \times B \times (N-1). \tag{3.35}$$

3.7. Codificação e decodificação *stream-oriented*

As descrições dos esquemas turbo dadas nas primeiras seções deste capítulo são suficientemente genéricas e servem também para o caso da codificação *stream-oriented*.

Basta considerar que o entrelaçador E na figura 3.1 é um entrelaçador periódico, ou mais especificamente, um entrelaçador convolucional.

O fragmento de grafo-fator do sistema de codificação turbo padrão dado na figura 3.4 também serve de referência para ilustrar as variáveis envolvidas e suas conexões. A diferença essencial reside nas conexões que o entrelaçador determina. Para um entrelaçador convolucional, que não se encerra em blocos, em princípio o grafo completo é infinito verticalmente para incluir as sequências completas de variáveis e conexões convolucionais.

Uma vez que os textos de referência dados na literatura descrevem bem a codificação *stream-oriented*, podemos nos concentrar em nossa real contribuição: uma descrição da decodificação contínua em termos de cronogramas do algoritmo SP.

Seguindo a mesma notação da literatura dada, os parâmetros que caracterizam o atraso total na decodificação são D, W e I, em que D é a latência essencial do entrelaçamento/desentrelaçamento definida nas seções anteriores, W é o comprimento gasto na apredizagem de estados na recursão *backward* , e o parâmetro I é o número fixo de iterações escolhido para a decodificação.

Na codificação contínua, não há na sequência de estados das treliças convolucionais estados forçados periodicamente (e, portanto, conhecidos pelo decodificador) como na codificação por blocos. Logo, na decodificação contínua, os cálculos sequenciais *forward* ou *backward*, que sempre partem de um ponto extremo para a recursão, não dispõem de informação perfeita sobre este estado no ponto extremo.

Costuma-se dizer que estes estados nas extremidades devem ser aprendidos. Essa aprendizagem se dá diferentemente, dependendo da modalidade: *forward* ou *backward*. Na modalidade *forward*, a estimativa do ponto extremo já está disponível, devido a cálculos anteriores na sequência, desde que o cronograma tenha sido projetado corretamente, como é o caso do apresentado aqui. Na modalidade *backward* , o estado extremo é desconhecido por princípio (por serem estados finais da sequência causal), mas sempre há o recurso de desconsiderar as estimativas do início do cálculo sequencial, os pontos perto desta extremidade. Usa-se, portanto, um comprimento de referência W de símbolos iniciais da

modalidade *backward* que servem para o aprendizado. Toda esta argumentação é inerente a cronogramas *forward-backward* em janela deslizante.

A diferença essencial na decodificação *stream* é o fato de que inevitavelmente, para um cronograma eficiente, tanto D quanto W se acumulam como atrasos sucessivos ao longo da realização das iterações do algoritmo. A latência D se acumula devido às operações de entrelaçamento/desentrelaçamento nas mensagens. Ou seja, estes atrasos se acumulam ao longo da arquitetura *pipeline* e são considerados inerentes à decodificação.

Assim, o atraso total de decodificação $\mathfrak{D}$ para I iterações é, conforme já estabelecido em [42] e [43],

$$\mathfrak{D} = I\,(D + 2W)\,. \tag{3.36}$$

Ou seja, $\mathfrak{D}$ aumenta linearmente com o número de iterações I. O termo $2W$ é consequência de termos duas treliças a realizar a *backward* .

3.7.1. O cronograma para a decodificação contínua convolucional

Conforme estabelecido pela generalização proveniente da teoria de grafos-fatores, a tarefa essencial de qualquer esquema de decodificação turbo iterativa é obter marginalizações aproximadas nas variáveis de interesse, para assim efetuar decisões. Essencialmente, descrever uma realização de decodificação neste ambiente é fornecer um cronograma eficiente de mensagens para o algoritmo SP.

A idéia de um algoritmo de decodificação que opera nas sequências contínuas de dados, com indexações infinitas ou semi-infinitas (sobre $\mathbb{Z}$ ou $\mathbb{Z}^+$) incluindo as todas as variáveis do grafo-fator, é por si só uma abstração que traz uma visão privilegiada das nuances de uma decodificação contínua.

O grafo-fator e a notação introduzida no início do capítulo proporciona esta extensão facilmente. Basta pensar na versão infinita do grafo representado na figura 3.3, em relação às sequências dispostas verticalmente, ou mesmo na versão compacta da figura 3.4. Na verdade, esta última é ainda mais apropriada, uma vez que é genérica e serve naturalmente para a

indexação infinita.

Baseado nas idéias já descritas no início do capítulo e aproveitando a formulação do grafo-fator normal/causal e mensagens categorizadas α e β desenvolvidas, apresentamos na tabela 3.3 a seguir um cronograma que define uma decodificação eficiente para sistemas turbo *stream-oriented* . O cronograma apresentado aqui é eficiente para qualquer esquema de entrelaçamento convolucional generalizado dado pela lei de permutação (3.28).

- para cada período $m = 0, 1, \ldots$
  - atribuir $i_{\text{mín}} = mN$ e $i_{\text{máx}} = mN + N - 1$;
  - calcular $\beta\left(u_i^0\right)$, $\beta\left(c_i^1\right)$, $\beta\left(c_i^2\right)$ para $i = mN, \ldots, mN + N - 1$;
  - para cada iteração de índice $n = 0, 1, \ldots, I - 1$:
    - atualiza $\alpha\left(u_i^1\right)$ para $i = i_{\text{mín}}, \ldots, i_{\text{máx}}$ (*a priori* para C_1);
    - atualiza $\alpha\left(s_i^1\right)$ para $i = i_{\text{mín}}, \ldots, i_{\text{máx}}$ (*forward* dentro de C_1);
    - atribui novo valor $i_{\text{mín}} = i_{\text{mín}} - W$;
    - atualiza $\beta\left(s_i^1\right)$ para $i = i_{\text{máx}}, \ldots, i_{\text{mín}}$ (*backward* dentro de C_1);
    - atribui novo valor $i_{\text{máx}} = i_{\text{máx}} - W$;
    - atualiza $\beta\left(u_i^1\right)$ para $i = i_{\text{mín}}, \ldots, i_{\text{máx}}$ (extrínsecas de C_1);
    - atualiza $\alpha\left(u_j^2\right)$ para $j = i_{\text{mín}}, \ldots, i_{\text{máx}}$ (*a priori* para C_2);
    - atualiza $\alpha\left(s_j^2\right)$ para $j = i_{\text{mín}}, \ldots, i_{\text{máx}}$ (*forward* dentro de C_2);
    - atribui novo valor $i_{\text{mín}} = i_{\text{mín}} - W$;
    - atualiza $\beta\left(s_j^2\right)$ para $j = i_{\text{máx}}, \ldots, i_{\text{mín}}$ (*backward* dentro de C_2);
    - atribui novo valor $i_{\text{máx}} = i_{\text{máx}} - W$;
    - atualiza $\beta\left(u_j^2\right)$ para $j = i_{\text{mín}}, \ldots, i_{\text{máx}}$ (extrínsecas de C_2);
    - atribui novo valor $i_{\text{máx}} = i_{\text{máx}} - D$;
  - ir para próxima iteração;
  - atualizar $\beta\left(x_i\right)$ e decidir x_i para os valores finais $i = i_{\text{mín}}, \ldots, i_{\text{máx}}$;
- ir para próximo período atribuindo $m = m + 1$;

Tabela 3.3: Primeira formulação do cronograma

Como a unidade fundamental no sistema é o período do entrelaçamento/desentrelaçamento , todas as operações são baseadas no período N do entrelaçador convolucional. A cada

período, N bits são decodificados. D é múltiplo de N. Embora não estritamente necessário, é conveniente que W seja também um múltiplo de N.

Pode-se considerar todo sistema sincronizado e que as variáveis com índice i "ocorrem" no instante i.

É natural a separação temporal em períodos de dados $\{0, \ldots, N-1\}$, $\{N, \ldots, 2N-1\}$, e assim por diante, tal que em um dado período, os dados do intervalo $\{mN, \ldots, mN+N-1\}$ chegando do canal e disponíveis para o decodificador. Esse é o m-ésimo período, com $m = 0, 1, \ldots$ para mantermos compatibilidade com índices referenciados em zero.

Apresentamos duas formulações alternativas para o cronograma. A primeira utiliza variáveis auxiliares deslizantes $i_{\text{mín}}$ e $i_{\text{máx}}$ para definir o intervalo de cálculos em cada etapa. A tabela 3.3 exibe esta formulação.

Evidentemente, há várias similaridades com o cronograma para o grafo-fator de um sistema turbo clássico. Portanto, seguimos uma descrição o mais próxima possível dos cronogramas já dados na seções anteriores.

Como nos cronogramas anteriores, as etapas finais a cada iteração concluída é a decisão nas variáveis x_i, que consistem na obtenção de $\beta(x_i)$ e decisões por $\operatorname{argmax} \alpha(x_i)\,\beta(x_i)$.

Em todas as linhas, seguem cálculos de N mensagens em intervalos de índices definidos pelas variáveis $i_{\text{mín}}$ e $i_{\text{máx}}$. Nesta formulação, $i_{\text{mín}}$ e $i_{\text{máx}}$ começam nos índices de tempo das observações do canal mais atuais, e terminam deslizando retroativamente até os índices de tempo das variáveis a serem tomadas decisões.

Considerando a codificação do sistema em uma sequência semi-infinita, com início em índices zerados, deve ficar subentendido que $i_{\text{mín}}$ e $i_{\text{máx}}$ de resultados negativos sempre são truncados a zero. Ou seja, as atribuições devem ser entendidas como $i_{\text{mín}} = \text{máx}\{0, i_{\text{mín}} - W\}$, $i_{\text{máx}} = \text{máx}\{0, i_{\text{máx}} - W\}$, $i_{\text{mín}} = \text{máx}\{0, i_{\text{mín}} - D\}$ e $i_{\text{máx}} = \text{máx}\{0, i_{\text{máx}} - D\}$.

Na segunda formulação dada na tabela 3.4, rescrevemos o cronograma, substituindo os valores correntes de $i_{\text{mín}}$ e $i_{\text{máx}}$ em cada etapa em função de parâmetros e variáveis globais.

Em função dos parâmetros N, W, D e das variáveis m e n. A variável $m = 0, \ldots$ indica o período, e $n = 0, \ldots, I-1$ indica em que iteração estamos.

- para cada período $m = 0, 1, \ldots$
  - calcular $\beta\left(u_i^0\right)$, $\beta\left(c_i^1\right)$, $\beta\left(c_i^2\right)$ para $mN \leq i < mN + N$;
  - para cada iteração de índice $n = 0, 1, \ldots, I-1$ atualizar as mensagens:
    - $\alpha\left(u_i^1\right)$, $mN - nD - (2n+0)W \leq i < mN - nD + N - (2n+0)W$
    - $\alpha\left(s_i^1\right)$, $mN - nD - (2n+0)W \leq i < mN - nD + N - (2n+0)W$
    - $\beta\left(s_i^1\right)$, $mN - nD - (2n+1)W \leq i < mN - nD + N - (2n+0)W$
    - $\beta\left(u_i^1\right)$, $mN - nD - (2n+1)W \leq i < mN - nD + N - (2n+1)W$
    - $\alpha\left(u_j^2\right)$, $mN - nD - (2n+1)W \leq j < mN - nD + N - (2n+1)W$
    - $\alpha\left(s_j^2\right)$, $mN - nD - (2n+1)W \leq j < mN - nD + N - (2n+1)W$
    - $\beta\left(s_j^2\right)$, $mN - nD - (2n+2)W \leq j < mN - nD + N - (2n+1)W$
    - $\beta\left(u_j^2\right)$, $mN - nD - (2n+2)W \leq j < mN - nD + N - (2n+2)W$
  - ir para próxima iteração, atribuindo $n = n + 1$;
  - decidir em x_i , $mN - I(D+2W) \leq i < mN - I(D+2W) + N$;
- ir para próximo período, atribuindo $m = m + 1$;

Tabela 3.4: Segunda formulação do cronograma

Em um dado período, temos os dados referentes ao canal com índices $mN \leq i < mN+N$ chegando, e as decisões ocorrendo nos índices no intervalo atrasado $mN - I(D+2W) \leq i < mN + N - I(D+2W)$.

Nesta segunda formulação fica mais compreensível como acontece o atraso ao longo das iterações, e precisamente em quais intervalos de índices devem ocorrer as atualizações.

Ao formular o cronograma em função de indexadores deste tipo, entendidas como variáveis que são controladas pelo receptor iterativo, fica mantida a generalidade da descrição, e ainda temos a indicação direta de implementação computacional por linguagem orientada a objetos.

Da forma como está projetado, o cronograma gerencia de forma ótima o processo de

apredizagem e o uso de estados nas extremidades dos cálculos *forward-backward*.

Além disso, diferentemente do esquema tradicional em *pipeline*, o cronograma dado restaura a idéia de apenas um conjunto decodificador. Basta portanto um *buffer* de memória para cada sequência de símbolos do grafo-fator, onde as mensagens α e β de toda a sequência são guardadas. Estas estimativas sobre a sequência serão melhoradas a cada iteração.

As unidades processadoras correspondem aos nós-função, que lêem e escrevem nos *buffers* adjacentes.

Todas as operações podem ser implementadas sobre *buffers* abstratos infinitos, que gerenciam a si próprios por alocações circulares (via álgebra módulo inteiro). Desta forma, devem ser criados os seguintes *buffers* associados X, U^0, U^1, U^2, S^1, S^2, C^1, C^2.

Cada *buffer* conserva as descrições das mensagens α e β das suas variáveis, que são vetores Q-ários de valores.

Sobre a demanda de memória para a realização de algoritmo em janela deslizante, os *buffers* circulares devem conservar uma janela de comprimento da ordem de $I\,(D+2W)$.

Note que usamos letras distintas i e j para as variáveis indexadoras das operações em cada um dos decodificadores 1 e 2. A finalidade é simplesmente evidenciar a diferenciação pelo entrelaçamento.

De fato, voltando à figura 3.6, podemos pensar nos nós-função das treliças centralizando os cálculos das mensagens nos ramos conectados a ele. Na figura, o nó-função relativo ao índice i na treliça de C_1 e o nó-função relativo a j na treliça de C_2 têm as respectivas funções associadas

$$p\left(c_i^1, s_{i+1}^1 | s_i^1, u_i^1\right) \text{ e } p\left(c_j^2, s_{j+1}^2 | s_j^2, u_j^2\right).$$

Podemos pensar em i como sendo a indexação global do sistema, e j como uma indexação sobre a influência do entrelaçamento.

Assim, ainda na figura 3.6, pensando em nós-funções como processadores ativos, o nó-função j lê a mensagem $\alpha\left(u_{\pi(j)}^2\right)$ (acessando o *buffer* de memória na posição dada pela lei de permutação), usa esta mensagem nos cálculos de $\alpha\left(s_{j+1}^2\right)$ e $\beta\left(s_j^2\right)$, e similarmente, calcula

e atualiza (escreve no *buffer*) a mensagem $\beta\left(u^2_{\pi(j)}\right)$. Esse direcionamento de mensagens fica transparente ao cronograma, que se preocupa em descrever apenas quais e quando as mensagens devem ser calculadas.

Dito isso, podemos adicionar mais alguns detalhes da situação transiente inicial, que diz respeito aos cálculos que envolvem as mensagens relativas ao início da treliça do codificador E_2 conectado ao entrelaçador.

Para alguns dos j iniciais no intervalo $j = 0, \ldots, D$ as mensagens $\alpha\left(u^2_{\pi(j)}\right)$ e $\beta\left(u^2_{\pi(j)}\right)$ se referem a índices negativos. Isto é, $\pi(j) < 0$ no início para alguns dos $j = 0, \ldots, D$. As primeiras podem ser lidas como deltas de Kronecker em zero (já que na codificação convolucional semi-infinita, considera-se todos os estados negativos zerados). E as segundas, simplesmente não serão usadas, e portanto não precisam ser calculadas e passadas.

3.7.2. Resultados de desempenho

A seguir, apresentamos alguns gráficos de desempenho obtidos por uma implementação computacional completa e geral do algoritmo *stream-oriented* e seu cronograma eficiente descrito na seção anterior. Algumas observações relevantes são feitas sobre a busca de parâmetros para bons desempenhos de sistemas deste tipo.

Seguimos na mesma linha dos resultados fornecidos na literatura consagrada no tema, alguns dos quais usamos como referência para comparação.

Porém, como contribuição, indicamos aqui algumas análises adicionais que julgamos relevantes, como o estudo da saturação do algoritmo em função do número de iterações. Essa análise é relevante pois no caso de entrelaçadores não-bloco a contagem do número de iterações I é parte integrante do atraso real de decodificação.

A tradução dos cronogramas dados aqui para uma implementação computacional por linguagem orientada a objetos é direta. Neste ambiente, é possível definir objetos abstratos, como cada sequência de variáveis no grafo-fator, seus índices como variáveis, e a função de permutação como uma função operando sobre estas "variáveis índices".

Em uma implementação maleável como foi realizada, todos os parâmetros de projeto do sistema podem ser escolhidos e fixados livremente, tais como o período N do entrelaçador, o parâmetro de multiplicidade B para entrelaçador convolucional linear, o comprimento W, número de iterações I, etc.

Podemos em uma primeira visão geral discutir valores típicos dos parâmetros empregados. Para W, em nossos experimentos concluímos que uma boa escolha costuma ser $W = N$. Com codificadores de memória 3 e 4, que são os usuais, e $N > 5$, isto é mais do que suficiente. Em alguns casos, até mesmo $W = 0$ não representa grande perda de desempenho.

Os valores N e B definem unicamente o entrelaçador convolucional da codificação, e a latência essencial D resultante. Seguimos a análise geral de [43] para suas escolhas. Na verdade, a maior parte das simulações apresentadas a seguir repetem os mesmos parâmetros de sistema empregados no artigo de referência. As curvas de desempenho desta seção representam a frequência relativa de erros de bits BER (*bit error rate*, BER) em escala logarítmica *versus* a relação sinal-ruído por bit de informação, denotada por SNR (o mesmo que E_b/N_0).

Um primeiro ponto de interesse é comparar o desempenho obtido pela decodificação proposta com os resultados da decodificação tradicional apresentados em [43]. Para isso, reproduzimos os mesmos parâmetros de sistema usados na referência. No que se refere aos codificadores convolucionais, sempre os mesmos geradores de paridade foram usados. Códigos de 8 ou 16 estados, com geradores convolucionais recursivos dados por $(1 + D + D^2 + D^3) / (1 + D^2 + D^3)$ e $(1 + D + D^3 + D^4) / (1 + D + D^4)$. Em notação octal para as conexões dos registradores de deslocamento, $(17/15)_8$ e $(33/31)_8$. Em relação às taxas do turbo, as mais comuns 1/2 (paridades puncionadas) ou 1/3.

As figuras 3.8 e 3.9 apresentam resultados de desempenho para conjunto de parâmetros idênticos a alguns simulados em [43]. Mais especificamente, nas suas figuras 15 e 14, respectivamente. Com estas comparações, podemos confirmar a eficiência de nosso cronograma e também verificar que a implementação computacional não tem falhas.

De fato, em geral as curvas obtidas aqui são semelhantes, mas com desempenhos

superiores aos de [43], que usou uma versão Max-Log-MAP do BCJR na decodificação.

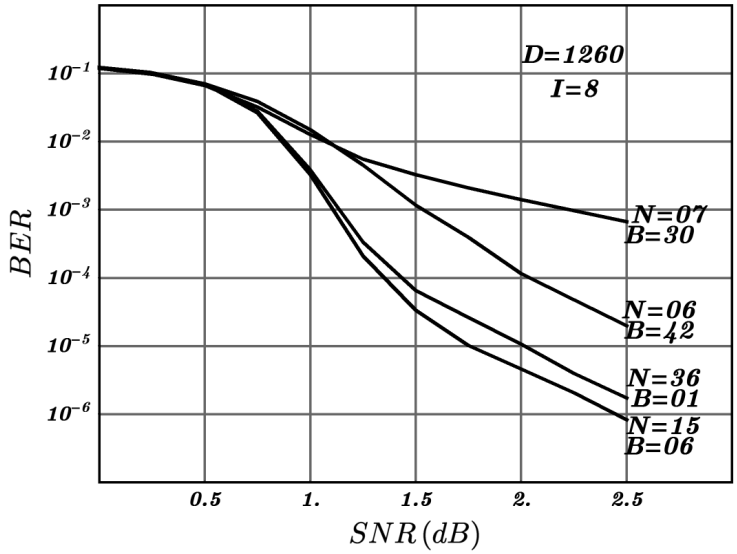

Figura 3.8: Desempenhos comparativos, variando o par de parâmetros (N, B) do entrelaçador, tal que $D = 1260$ é constante.

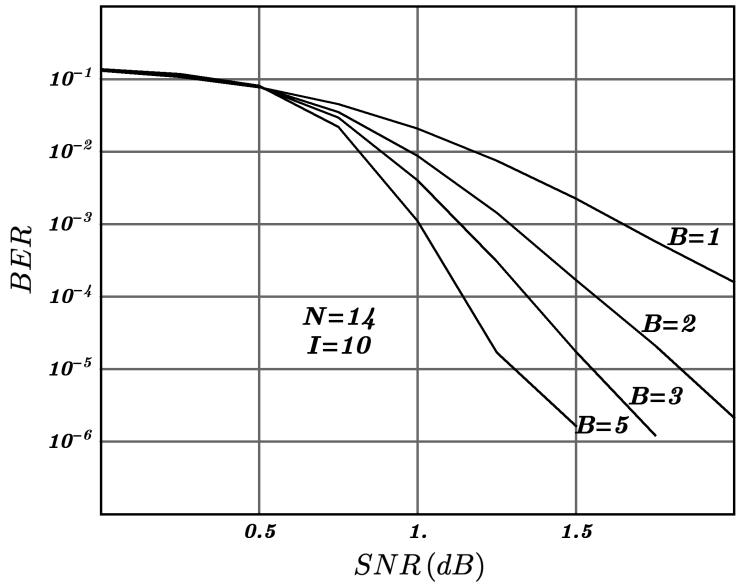

Figura 3.9: Desempenhos comparativos, variando o parâmetro B do entrelaçador.

A análise na figura 3.8 considera codificadores fixos de 8 estados e foca nas possíveis

variações dos parâmetros N e B do entrelaçador com a latência D constante, supostamente um limitante crítico real de interesse. Na figura 3.9, o foco central de análise é o efeito do parâmetro de multiplicidade B para ganhos de desempenho. Codificadores de 16 estados são usados e a taxa é $1/2$.

Como pontuado anteriormente, um entrelaçador $\pi_{N,B}$ é o equivalente a uma concatenação serial de B entrelaçadores $\pi_{N,1}$. Portanto, esta análise de ganhos em função do parâmetro B tem relevância por estudar essencialmente o efeito desta concatenação. Além disso, é interessante apresentar o estudo mais completo deste caso, considerando a saturação com o número de iterações I.

As figuras 3.10 e 3.11 mostram as curvas de desempenho variando o número de iterações, para $B = 3$ e $B = 5$, respectivamente. Aqui, vemos que de fato ocorreu a saturação, e $I = 10$ de fato é suficiente. Em sistemas com parâmetros N e B desta ordem (pequenos), este comportamento é característico e previsível, mesmo com codificadores de 16 estados.

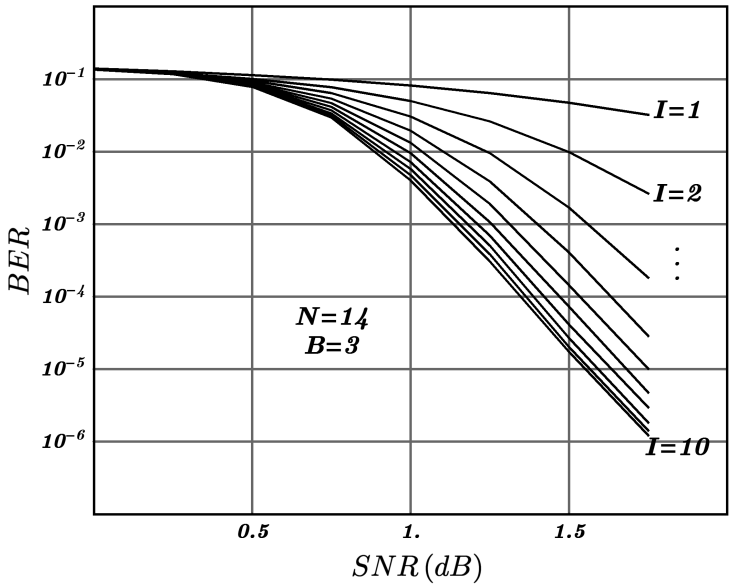

Figura 3.10: Estudo da saturação de desempenho de acordo com número de iterações. Sistema convolucional típico com $N = 14$ e $B = 3$.

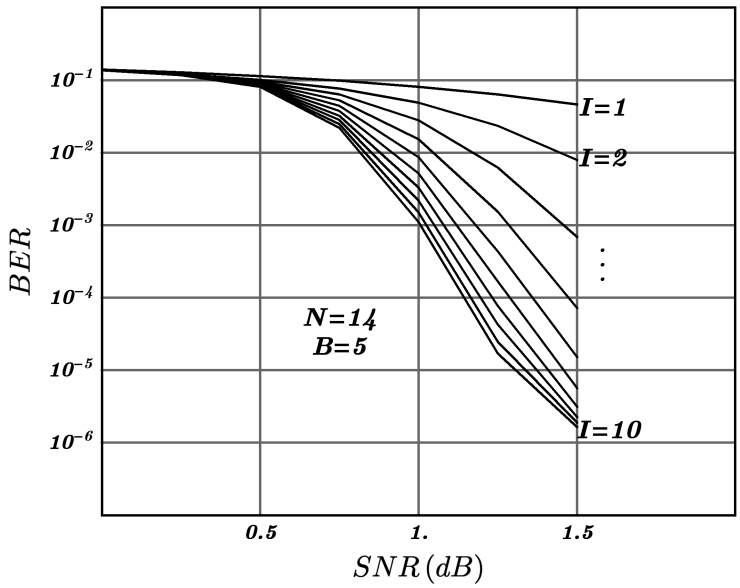

Figura 3.11: Estudo da saturação de desempenho de acordo com número de iterações. Sistema convolucional típico com $N = 14$ e $B = 5$.

O fato de que o atraso total de decodificação aumenta em função dos três parâmetros N, B e I nos leva a concluir que a busca por parâmetros para sistemas *stream-oriented* visando baixo atraso total e bom desempenho em geral devem considerar estas três variáveis conjuntamente, e não apenas N e B como é usualmente feito na literatura. Mais do que isso, para valores de I fixados na região conveniente (digamos $8 \leq I \leq 18$), deve ser ainda feito o estudo da saturação do desempenho em função do incremento em potencial de I que se pode intercambiar com N e B, mantendo o mesmo atraso.

A figura 3.12 é um exemplo deste tipo de análise. Nela, consideramos propositalmente conjuntos de parâmetros N, B, I para obter curvas de diversos desempenhos, mas todos com mesmo atraso total de decodificação $\mathfrak{D} = I\left(D + 2W\right) = I\left(N\left(N-1\right)B + 2W\right)$. Aqui, fizemos $W = 0$ em todas as configurações sem significativas perdas. Nestas simulações consideramos sistemas turbo contínuo de taxa $1/3$ (não puncionado) e geradores de paridade de 16 estados com $\left(1 + D + D^3 + D^4\right) / \left(1 + D + D^4\right)$.

O conjunto da curva (f) com parâmetros $N = 13, B = 7, I = 15$ resulta no melhor desempenho entre todos, indicando que em um de projeto onde a limitação crítica é de

atraso, pode ser interessante escolher menores valores de N e B para se permitir aumentar o número de iterações I, melhorando o desempenho. Ou seja, essencialmente intercambiar a grandeza de D pela grandeza de I.

Numa análise isolada, sem considerar I como parâmetro fundamental de desempenho, poderíamos nos limitar a enxergar apenas as curvas (c) (d) e (e), todas com mesmo D, sendo que a curva (f) supera em desempenho todas estas, com menor D.

Um estudo mais apropriado é observar se desempenhos de diferentes pares de N e B são melhores ou piores com relação ao potencial de ganho com o incremento de iterações. Tomando a comparação entre as duas curvas (a) e (f), uma com o menor I e a outra com o maior I empregado entre todas, podemos observar nas figuras 3.13 e 3.14 a saturação em relação a I.

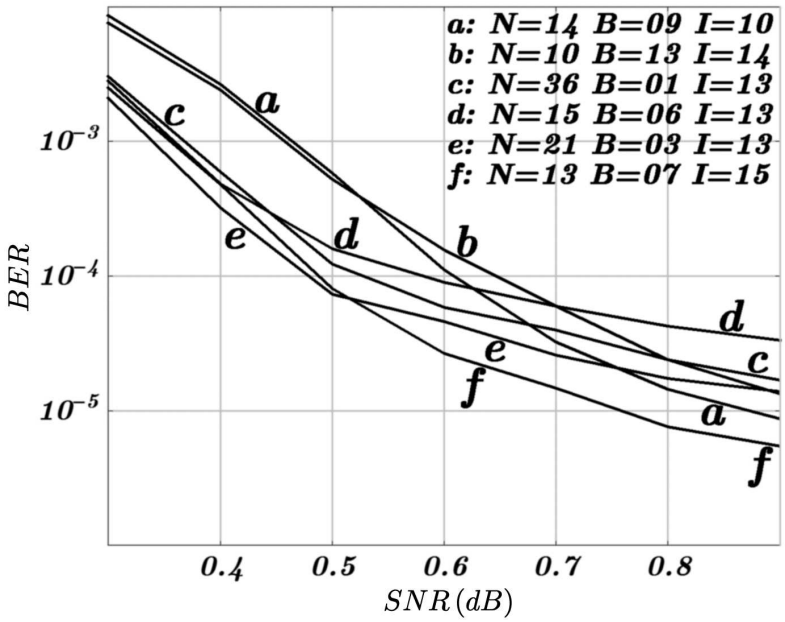

Figura 3.12: Curvas de desempenho para diversos conjuntos (N, B, I) com mesmo atraso de decodificação.

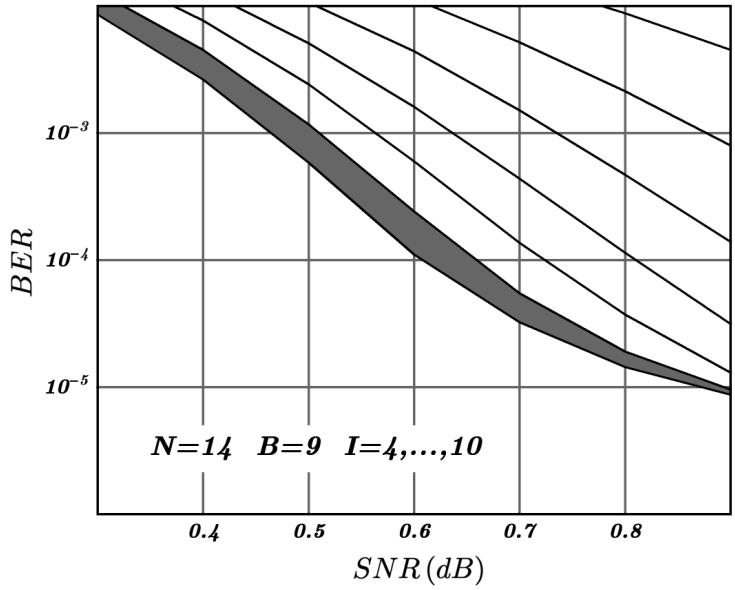

Figura 3.13: Estudo da saturação de desempenho em relação a I, até o limite $I = 10$ da curva (a) da figura 3.12.

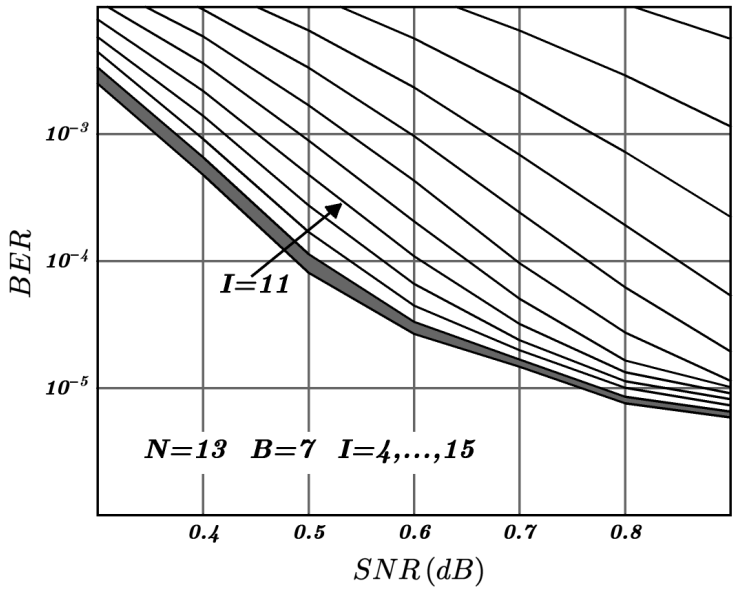

Figura 3.14: Estudo da saturação de desempenho em relação a I, até o limite $I = 15$ da curva (f) da figura 3.12.

Nas duas figuras, as regiões sombreadas representam o ganho de desempenho na última

iteração antes do limite (imposto pelo atraso $\mathfrak{D}$).

As diferenças significativas de desempenho entre estas duas configurações ocorrem na região intermediária dos gráficos. Na figura 3.13, vemos que embora em quase toda a região E_b/N_0 destacada haja um ganho significativo na última iteração, o ponto extremo $E_b/N_0 = 0{,}9dB$ está perto de saturação. Uma saturação de mesma ordem da que ocorre na figura 3.14, mas para um I maior. Conforme indicado nesta figura, vemos que para I similar ao saturado da primeira ($I = 11$), os desempenhos e potenciais de ganho são similares nas duas configurações. A diferença é realmente a possibilidade de se chegar até $I = 15$ obedecendo ao limite $\mathfrak{D} = 16380$.

Assim, mostramos através destes exemplos que no estudo geral de desempenho da decodificação quando considerado o atraso total $\mathfrak{D}$ e não simplemente a latência D (inerente apenas ao entrelaçamento de codificação), surge a necessidade de uma análise mais complexa, mas que é realista sobre os verdadeiros parâmetros críticos de um sistema *stream-oriented*. A descrição destes exemplos servem de indicação das direções a seguir para uma análise deste tipo. Municiado da implementação computacional eficiente que o cronograma descrito anteriormente proporciona, estão dadas as condições para isso.

3.8. Conclusões gerais sobre o capítulo

Neste capítulo, relacionamos de forma inovadora a teoria de codificação e decodificação turbo com teoria de grafos-fatores e o algortimo genérico SP. Através de uma realização de grafo-fator bem estruturada, foram descritas de forma mais apropriada todas as variáveis envolvidas em um esquema teórico de transmissão turbo padrão, que conserva a idéia geral da marginalização no algoritmo SP como o fundamento básico envolvido na decodificação. A realização conserva uma estruturação de causalidade entre as variáveis do sistema, o que traz vantagens no seu entendimento heurístico. Ao mesmo tempo, o cuidado em obter um grafo-fator normalizado resulta em uma simplificação interessante na descrição das mensagens típicas do algoritmo SP. As mensagens são categorizadas, relacionando seu direcionamento

com a direção de causalidade do sistema. Mais uma vantagem heurística.

Assim como a teoria geral de grafos-fatores trouxe uma nova forma elegante de descrição de sistemas e suas respectivas análises de marginalização, consideramos que a derivação desenvolvida aqui pode ser implementada de forma análoga para inúmeros sistemas descritos por grafos-fatores, representando um pequeno passo adiante no sentido de simplificar entendimentos. A segunda parte do capítulo é um exemplo direto de como a sua aplicação pode trazer contribuições.

Proporcionalmente, esta derivação também simplifica o desenvolvimento de programas computacionais abstratos para simular sistemas gerais de decodificação iterativa e como consequência até mesmo implementações reais de decodificadores iterativos. De posse de uma implementação computacional geral e abstrata que foi desenvolvida em nosso projeto, pudemos simular inúmeros casos de sistemas de interesse. Ao longo do texto, alguns resultados de simulação obtidos são apresentados.

A sequência do capítulo se dedica a descrever nossa proposta de redescoberta do paradigma turbo contínuo (ou turbo *stream-oriented*) utilizando a nossa derivação, que novamente provou sua utilidade. Antes disso, como fundamentação prévia dos conceitos envolvidos, foi apresentada toda a parte da teoria geral de entrelaçadores relevante para a descrição deste paradigma.

O objetivo central da segunda parte foi apresentar uma formulação de algoritmo (cronograma) de decodificação inovadora que desenvolvemos para o paradigma turbo *stream-oriented* . Nossa formulação possui vantagens de implementação, como melhor gerenciamento de processamento e memória em relação à arquitetura proposta em [43].

Na literatura existente sobre o tema turbo *stream-oriented* , a decodificação *stream-oriented* é apenas esquematizada sem maiores detalhes como uma extensão natural derivada da decodificação turbo clássica. Isso implica em um esquema potencialmente dispendioso, com a necessidade de múltiplos módulos de processamento e múltiplos módulos de memória.

Acreditamos que este fato contribuiu para a repercussão relativamente pequena deste paradigma, muito embora mostrasse vantagens comparativas convincentes em relação ao turbo clássico para sistemas reais com limitação de atraso e necessidade de baixa ambiguidade de sincronismo.

O cronograma eficiente aqui descrito resolve de forma definitiva este problema, restaurando a idéia de uma decodificação sem multiplicidade de módulos. Além disso, tanto sua implementação computacional para fins de simulações como sua implementação prática em sistemas reais são factíveis, com eficiência e generalidade.

Finalmente, a última parte do capítulo apresenta alguns resultados de simulações obtidas no intuito de exemplificar o tipo de estudo e dilemas intrínsecos a um sistema turbo *stream-oriented*, seus parâmetros e típicos compromissos que refletem no desempenho. Nesta parte, ampliamos a análise da literatura existente em relação ao estudo do atraso total de decodificação, indicando a necessidade do estudo sobre a saturação com iterações.

Capítulo 4

Decodificação conjunta iterativa vetorial

O objetivo deste capítulo é apresentar o estudo de uma instância do problema geral de transmissão digital de uma fonte contínua em amplitude, utilizando a teoria de grafos-fatores como ferramenta. Em particular, como modelar o sistema de tal forma que ele admita decodificação iterativa conjunta fonte-canal. Partimos de um modelo genérico de fonte, um processo estocástico discreto no tempo e contínuo em amplitude, que passa por quantização vetorial e indexação binária, e os bits resultantes são adicionalmente protegidos por codificação para uma transmissão através de um canal ruidoso. A idéia é fornecer uma visão geral de como todos estes elementos podem ser modelados para uma decodificação iterativa conjunta, com um enfoque não encontrado na literatura. As principais referências com as quais nosso estudo se relaciona são [1], [13], [14], [22], [34], [65] e principalmente [36].

Através da teoria de grafos-fatores, é vislumbrada uma completa caracterização do que são as variáveis do sistema e suas inter-relações, tudo sendo sintetizado na fatoração de um função custo conjunta entre todas elas, que eventualmente seria a distribuição conjunta entre as variáveis, tal que uma decodificação por marginalização aproximada se torna realizável pelo algoritmo SP iterativo em um grafo com ciclos.

Assim, vários conceitos são abordados, tais como decodificação com grafos-fatores

envolvendo variáveis contínuas, modelos para a fonte baseados em processos estocásticos discretos lineares, incorporando-os aos grafos-fatores da rede de decodificação proposta, quantização vetorial e mapeamento binário com seus mapas inseridos no grafo-fator do sistema. Ao final, apresentamos resultados simulados para decodificação iterativa conjunta.

4.1. Introdução

O princípio da separação de Shannon estabelece que é possível codificar fonte e canal separadamente com otimalidade, desde que a entropia da fonte H não exceda a capacidade do canal C por onde os dados devem ser transmitidos. Para provar o resultado, Shannon utiliza um método assintótico, que sugere a codificação sendo feita em blocos de dimensão não-limitada, implicando em complexidade e atraso potencialmente infinitos.

A idéia de decodificação conjunta vem do fato de que para sistemas realistas de comunicação digital, o teorema de Shannon da separação entre codificação-fonte e codificação-canal com suas hipóteses ideais não pode ser aplicado. Ele enuncia que dada uma fonte e um canal compatíveis (em termos de distorção e taxa) para transmissão, o problema de obter um esquema ótimo de codificação pode ser completamente separado em um esquema de codificação/decodificação ótima para a fonte, e outro esquema de codificação/decodificação ótima para o canal, desde que o comprimento de bloco não seja restrito. Isso não é possível na prática, com comprimentos finitos. Em alguns casos, devido à restrição de atraso de transmissão, até comprimentos mínimos são requeridos. Além disso, raramente temos esquemas ótimos para cada um deles mesmo quando pensados separadamente.

Podemos definir o termo **recurso de codificação conjunta** fonte-canal como sendo qualquer advento ao longo da cadeia de codificação que vise minimizar a distorção final da reprodução da fonte no seu destino, e que envolva o projeto combinado das duas partes: codificação de fonte e codificação de canal. Similarmente, podemos nos referir ao termo **recurso de decodificação conjunta** fonte-canal, que pensado conjuntamente com recursos de codificação conjunta ou assumindo um projeto de codificação dado, visa melhorar o desempenho final da transmissão.

Entre os recursos de codificação conjunta, podemos citar alguns consagrados, atuando nos mais diversos componentes típicos de uma codificação. Recursos de quantização robusta, tais como COVQ (*Channel Optimized Vector Quantization*) [55], [22], [23], [24] e CCVQ (*Channel Constrained Vector Quantization*) [78]. Recursos de otimização do mapeamento entre índices do quantizador e os vetores de bits a serem transmitidos (*Index Assignment*, IA), tratados em [22], [23], [18], [19], e principalmente [89], onde é apresentado o algoritmo BSA (*Binary Switch Algorithm*) que obtém mapeamentos localmente ótimos por permutações simples.

O estudo de questões sobre mapeamento binário é uma vasta linha de pesquisa. Uma extensão relevante para nosso contexto são os mapeamentos com redundância adicional (*Redundant Index Assignment*, ou RIA), discutidos em [1], [13], [70] e [14].

Outro recurso amplamente utilizado é a proteção desigual de erros (*Unequal Error Protection*, UEP), introduzida desde [62], e redesenhada em [40] para códigos convolucionais.

Portanto estes são exemplos variados de recursos de codificação conjunta, atuando em diferentes componentes do esquema acima.

Já em relação aos **recursos de decodificação conjunta**, podemos destacar duas principais linhas de desenvolvimento precursoras. O recurso SCCD (*Source-Controlled Channel Decoding*) desenvolvido em [41][49][47][48], e o recurso SBSD (*Softbit Source Decoding*) desenvolvida em [25][26][27]. A decodificação iterativa fonte/canal apresentada em [36] pode ser considerada uma sobreposição a estas duas, pois englobou e aperfeiçoou ambas. É atribuída a sigla ISCD (*Iterative Source-Channel Decoding*) ao método apresentado em [36].

Outro recurso de decodificação conjunta comum é a utilização da decisão por mínima distorção quadrática (MMSE, *Minimum Mean Square Error*), tomando médias ponderadas de centróides da quantização.

De forma geral, a codificação de fonte em esquemas realizáveis quase sempre não é ótima, e temos algum nível residual de redundância na saída do seu codificador. A idéia básica é utilizar esta redundância residual para ajudar o decodificador de canal a eliminar erros. Daí vem toda uma linha de técnicas de projeto de sistemas onde esta redundância é controlada

e modelada na codificação, para que possa ser útil em uma decodificação turbo sub-ótima.

4.2. Esquema de codificação e decodificação conjunta

Consideramos o problema geral de transmissão digital de uma fonte discreta no tempo e contínua em amplitude (valores em $\mathbb{R}$). Partimos de um esquema geral de comunicação, com modelo usual de codificação de fonte e codificação de canal.

A fonte é quantizada vetorialmente. Os índices, para a finalidade de uma transmissão digital são associados a vetores binários. Este mapeamento entre índices e bits (*bit assignment*) é considerado como elemento à parte no esquema de codificação.

Consideramos uma quantização vetorial projetada com critérios para mínima distorção separadamente (assumindo transmissão sem erro) pelo GLA (*Generalized Lloyd Algorithm*), e posteriormente projetando a indexação binária para canal ruidoso, pelo BSA (*Binary Switch Algorithm*).

O sistema é ilustrado na figura 4.1, representado analiticamente de forma conveniente pelos seguintes componentes: fonte, quantizador vetorial, mapeador entre vetores e palavras binárias (*bit assignment*), codificador de canal e canal.

Figura 4.1: Esquematização do sistema para decodificação conjunta.

A fonte emite símbolos X_l $(l = 0, 1, \ldots)$, um processo aleatório estacionário discreto no tempo e contínuo em amplitude. Esse processo é separado em blocos de m símbolos e quantizado vetorialmente por uma aplicação $\psi : \mathbb{R}^m \rightarrow \{0, \ldots, n-1\}$, que resulta na sequência Q_j $(j = 0, 1, \ldots)$, um processo discreto com alfabeto $\{0, \ldots, n-1\}$. Acoplado ao quantizador, consideramos o mapeamento injetor para associar índices $\{0, \ldots, n-1\}$

a palavras binárias de comprimento s, ou seja, elementos em $\{0,1\}^s$ (taxa resultante do sistema de quantização é s/m bits por amostra). Eventualmente, este mapeamento pode adicionar redundância para proteção contra erros. Denotamos por B_k a sequência de bits emitidos serialmente pelo sistema de quantização. Um codificador toma a sequência de bits B_k $(k = 0, 1, \ldots)$ como entrada e emite serialmente a sequência de bits codificados C_t $(t = 0, 1, \ldots)$ sinalizados como $\{-1, +1\}$ para um canal B-AWGN. Y_t é a sequência de observações ruidosas na saída do canal. O objetivo final do sistema de codificação e decodificação é transmitir o processo X_l $(l = 0, 1, \ldots)$ com a mínima distorção possível. A seguir, alguns detalhes sobre cada um destes componentes.

A idéia é associar um componente decodificador a cada um dos componentes do sistema de codificação dado acima entre a fonte e o canal (quantizador, mapeamento binário e código de canal). Ao construir o grafo-fator do sistema, onde as variáveis envolvidas em cada um deles são propriamente conectadas, vem naturalmente a visualização de como se realiza a decodificação conjunta por marginalizações com o algoritmo SP.

O princípio turbo de decodificação é a iteração entre dois (ou mais) componentes de um sistema de decodificação (dual ao sistema de codificação). Cada um deles melhora as estimativas das variáveis envolvidas utilizando os modelos de memórias e redundâncias implícitas no sistema. Em princípio, uma fonte com memória deve participar deste processo.

4.2.1. A fonte

Um modelo comum para uma fonte genérica é um processo que emite símbolos discretos no tempo e contínuos na amplitude, com alguma propriedade de estacionariedade. Por simplicidade, vamos nos restringir a processos auto-regressivos e/ou médias-móveis (processos ARMA) definidos a seguir. Podemos destacar o tipo mais simples deles, os processos de Gauss-Markov gerados pela equação

$$X_{t+1} = \alpha X_t + W_t, \tag{4.1}$$

em que W_t é um processo gaussiano de média zero e variância constante, com $\alpha < 1$. São processos caracterizados por apenas um parâmetro. O parâmetro α representa o grau

de determinismo do processo. Para processos deste tipo, a função de auto-correlação é simplesmente

$$E\left[X_t X_{t-s}\right] = \alpha^s \tag{4.2}$$

e, portanto, tem decaimento exponencial.

Processos estocásticos ARMA são processos derivados a partir de processos de ruído branco, através de combinações lineares de janelas temporais.

Um **ruído branco** é qualquer processo cuja sequência temporal são variáveis identicamente distribuídas, de média zero e não correlacionadas. No que se segue, E_t $(t = 0, 1, \ldots)$ denota um ruído branco com alguma distribuição fixa, de variância σ^2.

Um processo X_t é dito **estacionário fraco** quando seus primeiros e segundos momentos são invariantes por translação no tempo, isto é, $E\left[X_t\right]$ e $E\left[X_t^2\right]$ são constantes em t, e $E\left[X_t X_{t-s}\right]$ só depende de s. No que se segue, consideramos a propriedade de estacionariedade apenas no sentido fraco.

Ruídos brancos são casos particulares de processos estacionários em que $E\left[X_t\right] = 0$ e $E\left[X_t X_{t-s}\right] = 0$ para $s > 0$. Os casos mais comuns de ruído branco são gaussianos, embora esta restrição não seja necessária em geral.

Um processo X_t é dito M.A. (*moving average*) de ordem p, se pode ser escrito como

$$X_t = E_t + \theta_1 E_{t-1} + \ldots + \theta_p E_{t-p} \ , \tag{4.3}$$

onde $\theta_1, \ldots, \theta_p$ são coeficientes constantes. Processos deste tipo são sempre estacionários, por serem uma combinação linear de processos estacionários e de covariâncias nulas. Além disso, temos $E\left[X_t^2\right] = \sigma^2\left(1 + \theta_1^2 + \ldots + \theta_p^2\right)$ e $E\left[X_t X_{t-s}\right] = 0$ para $s > p$ (correlação limitada no tempo).

Um processo Y_t é dito A.R. (*auto regressive*) de ordem r, se pode ser escrito como

$$Y_t = \phi_1 Y_{t-1} + \phi_2 Y_{t-2} + \ldots + \phi_r Y_{t-r} + E_t \ , \tag{4.4}$$

onde $\phi_1, \ldots, \phi_r$ são coeficientes constantes. Um processo A.R. é estacionário se e só se o polinômio $1 - \phi_1 x - \phi_2 x^2 - \ldots - \phi_r x^r$ tem todas as suas raízes (complexas ou reais) fora do círculo unitário de $\mathbb{C}$, ou seja, $1 - \phi_1 x - \phi_2 x^2 - \ldots - \phi_r x^r = 0 \Rightarrow |x| > 0$.

Em particular, um processo A.R. de primeira ordem é estacionário se seu coeficiente θ_1 é menor que 1. Logo, os processos de Gauss-Markov definidos na seção anterior são estacionários por construção.

Um processo ARMA de ordem (r, p) é um processo da forma

$$Y_t = \phi_1 Y_{t-1} + \phi_2 Y_{t-2} + \ldots + \phi_r Y_{t-r} + E_t + \theta_1 E_{t-1} + \ldots + \theta_p E_{t-p} \; . \tag{4.5}$$

Evidentemente, processos A.R. e processos M.A. são casos particulares de processos ARMA

A estacionariedade de um processo ARMA só depende de sua parte auto-regressiva (similar aos processos A.R.), sendo estacionário se e somente se o polinômio $1 - \phi_1 x - \phi_2 x^2 - \ldots - \phi_r x^r$ tem todas as suas raízes fora do círculo unitário de $\mathbb{C}$.

4.2.2. Quantização vetorial

Um resultado fundamental da teoria de taxa-distorção para codificação de fonte é que a quantização vetorial traz ganhos em relação à quantização escalar. Sobretudo quando as amostras são estatisticamente dependentes. Nos casos aqui considerados, para dimensões modestas, o ganho em resolução é compensador, na medida em que as incertezas do canal são resolvidas pela decodificação iterativa conjunta.

No que se segue, apresentamos em linhas gerais os conceitos de um quantizador vetorial da categoria mais simples (sem memória), e o algoritmo usual de obtenção prática de quantizadores deste tipo com critério de otimalidade quando o codificador de fonte é projetado isoladamente, ou ainda, admitindo transmissão perfeita, sem ruído.

Consideramos um quantizador vetorial em $\mathbb{R}^m$, caracterizado por um mapa $\psi : \mathbb{R}^m \to I$ que associa a cada vetor em $\mathbb{R}^m$ um elemento em um conjunto finito de índices I e um mapa inverso unívoco $\varphi : I \to \mathbb{R}^m$, que associa a cada índice $i \in I$ o seu respectivo vetor código $\mathbf{z}_i \in \mathbb{R}^m$. O quantizador vetorial é definido pela composição das duas funções, $\chi = \varphi \circ \psi$.

A imagem de χ, a mesma de φ, é o dicionário do quantizador $\{\mathbf{z}_i = \varphi(i) : i \in I\}$.

Associada ao mapa ψ existe a partição natural, a classe de subconjuntos disjuntos suplementares em $\mathbb{R}^m$ dada pela imagens inversas $\left\{S_i = \psi^{-1}(i) : i \in I\right\}$. Podemos dizer que S_i é a "região" ou "partição" associada a $\mathbf{z}_i$.

De forma equivalente, o quantizador vetorial pode ser caracterizado definindo-se duas entidades da teoria de conjuntos: uma partição finita $\{S_i \subseteq \mathbb{R}^m : i \in I\}$ do espaço $\mathbb{R}^m$, e um conjunto finito de elementos $\{\mathbf{z}_i \in \mathbb{R}^m : i \in I\}$ associados.

Dada uma medida de distorção $d : \mathbb{R}^m \times \mathbb{R}^m \rightarrow \mathbb{R}^+$ e sendo $p(\mathbf{x})$ a densidade de probabilidade em $\mathbb{R}^m$ relativa à estatística da fonte, um quantizador χ é ótimo em relação a d e p se ele minimiza a distorção média

$$\int_{\mathbf{x}\in\mathbb{R}^m} d(\mathbf{x}, \chi(\mathbf{x}))\, p(\mathbf{x})\, d\mathbf{x} \tag{4.6}$$

No que se segue, consideramos a distorção quadrática $d(\mathbf{x}, \mathbf{z}) = \|\mathbf{x} - \mathbf{z}\|^2$ em $\mathbb{R}^m$. O desempenho da quantização é usualmente avaliado pela relação sinal-ruído de quantização,

$$SQNR = 10 \log_{10} \frac{\int_{\mathbf{x}\in\mathbb{R}^m} \|\mathbf{x}\|^2 p(\mathbf{x})\, d\mathbf{x}}{\int_{\mathbf{x}\in\mathbb{R}^m} \|\mathbf{x} - \chi(\mathbf{x})\|^2 p(\mathbf{x})\, d\mathbf{x}}. \tag{4.7}$$

As duas propriedades básicas atribuídas a qualquer quantizador ótimo são:

1. Para todo $\mathbf{x} \in \mathbb{R}^m$, o seu vetor código associado é o mais próximo, isto é

$$\chi(\mathbf{x}) = \text{argmin}_{\mathbf{z}_i}\, d(\mathbf{z}_i, \mathbf{x}). \tag{4.8}$$

2. Vetores código coincidem com os centróides das suas regiões, isto é, para cada $\mathbf{z}_i$

$$\mathbf{z}_i = \frac{\int_{S_i} \mathbf{x}\, p(\mathbf{x})\, d\mathbf{x}}{\int_{S_i} p(\mathbf{x})\, d\mathbf{x}}. \tag{4.9}$$

Estas são as duas condições necessárias para otimalidade, isto é, se um quantizador é ótimo e minimiza a distorção média dada por (4.6), então isto implica que ele satifaz (4.8) e (4.9).

O algoritmo básico para a obtenção de quantizadores vetoriais ótimos é o chamado algoritmo GLA (*Generalized Lloyd Algorithm*), conforme estabelecido em [58]. O algoritmo

utiliza as duas condições necessárias (4.8) e (4.9) para a obtenção iterativa de um quantizador ótimo.

Basicamente, o algoritmo opera sobre uma sequência de treinamento gerada conforme $p(\mathbf{x})$. Partindo de um dicionário inicial, iterativamente atualiza-se este dicionário usando as duas propriedades acima, aplicadas na sequência. A idéia é aglomerar os elementos mais próximos a cada vetor código, e usar esta aglomeração para reajustá-lo. Ou seja, a cada iteração: usa-se a propriedade (4.8) para particionar a sequência em subconjuntos (obtendo as aglomerações); e depois usa-se (4.9) para definir os novos vetores código como sendo os centróides dos subconjuntos (vetor médio de cada aglomeração).

Há métodos eficientes de escolha dos vetores código iniciais para o algoritmo GLA. O mais reconhecido é dado também em [58]. O algoritmo GLA provido deste método é o denominado algoritmo LBG (Linde-Buzo-Gray), que funciona para obtenção de dicionários de cardinalidade 2^n.

Em linhas gerais, este método consiste em realizar o algoritmo em múltiplas etapas (n etapas), em cada uma delas realizando um GLA para obter dicionários de $1, 2, 4, , \ldots, 2^n$ elementos. Dado um dicionário com 2^h elementos, é obtido um dicionário com 2^{h+1} por *splitting*, isto é, perturbações em cada uma das palavras para a obtenção de dois pontos próximos. Assim, a partir de 2^h, surgem $2 \times \left(2^h\right)$ vetores que podem ser usados como vetores código iniciais para uma nova etapa. Na primeira etapa, o dicionário de 1 elemento é dado pelo centróide da sequência inteira.

A quantizadores vetoriais assintoticamente ótimos obtidos por este algoritmo é associada a nomenclatura "quantizadores LBG". Em nosso sistema e em nossas simulações, consideramos sempre quantizadores LBG.

4.2.3. O quantizador vetorial

Dado o quantizador vetorial obtido conforme descrito na seção anterior com seus mapas associados ψ e φ, voltamos a considerar o seu papel no sistema proposto. Podemos convencionar para os índices o conjunto finito $I = \{0, \ldots, n-1\}$, tal que a taxa resultante

do quantizador é $\log_2 n/m$ bits por símbolo.

Como componente do esquema proposto, o quantizador vetorial gera a partir do processo contínuo $X_l \in \mathbb{R}$ $(l = 0, 1, \ldots)$ um processo discreto, a sequência temporal de índices $Q_j \in I$ $(j = 0, 1, \ldots)$.

Para quantização da sequência de um processo X_l, consideramos o particionamento em blocos de m símbolos consecutivos do tipo $(X_{jm}, \ldots, X_{jm+m-1})$, sendo emitido o índice $Q_j \in I$, dado pelo mapeamento

$$Q_j = \chi\left(X_{jm}, \ldots, X_{jm+m-1}\right) \tag{4.10}$$

Como X_l é um processo com memória, o processo discreto Q_j também possui algum tipo de memória. Quando X_l é um processo ARMA de ordem pequena, menor que a dimensão do quantizador $(p < m)$, uma boa aproximação é assumir que Q_j é um processo de Markov de primeira ordem estacionário, e estimar com sequências de treinamento suas probabilidades

$$P\left(Q_j = s, Q_{j+1} = t\right)_{s\in I, t\in I} \tag{4.11}$$

invariantes em j. O processo consiste em simular o processo X_l, obter o processo Q_j correspondente, e registrar as frequências relativas. A partir de (4.11), são naturalmente obtidas

$$P\left(Q_j = s\right)_{s\in I} \tag{4.12}$$

e as probabilidades de transição

$$P\left(Q_{j+1} = t | Q_j = s\right)_{s\in I, t\in I}. \tag{4.13}$$

A partir das distribuições, as entropias relevantes $H\left(Q_j\right)$ e $H\left(Q_{j+1}|Q_j\right)$ podem ser obtidas. A entropia $H\left(Q_{j+1}|Q_j\right)$ pode ser obtida por

$$H\left(Q_{j+1}|Q_j\right) = H\left(Q_j, Q_{j+1}\right) - H\left(Q_j\right). \tag{4.14}$$

A importância da probabilidade condicional $P\left(Q_{j+1} = t|Q_j = s\right)$ é servir como função custo para a decodificação *forward-backward* do processo Q_j. Passando para a notação de

grafos-fatores, denotamos simplesmente $p\left(q_{j+1}|q_j\right)$, onde os argumentos definem as variáveis nas quais a função está definida.

Recapitulando, por definição a entropia do processo é

$$\lim_{J\to\infty}\frac{1}{J}H\left(Q_1,\ldots,Q_J\right). \tag{4.15}$$

Assumindo a estacionariedade do processo, vale a igualdade

$$\lim_{J\to\infty}\frac{1}{J}H\left(Q_1,\ldots,Q_J\right). = H\left(Q_{j+1}|Q_j\right). \tag{4.16}$$

Portanto, a quantização do processo X_l resulta no processo Q_j, onde a entropia é reduzida, e $H\left(Q_{j+1}|Q_j\right)$ é uma aproximação desta entropia resultante.

Evidentemente, para considerar a taxa de entropia por símbolo da fonte na saída do quantizador, devemos considerar o ajuste $H\left(Q_{j+1}|Q_j\right)/m$.

Exemplo: para X_l um processo Gauss-Markov com coeficiente $\alpha = 0{,}95$, obtendo quantizadores LBG de dimensões $m = 3, 4, 5, 6$ e taxa de 1 bit/símbolo, as entropias resultantes são dadas na tabela 4.1.

m	3	4	5	6
$H\left(Q_j\right)/m$	0,94244	0,96850	0,97283	0,98029
$H\left(Q_{j+1}|Q_j\right)/m$	0,60193	0,68951	0,73946	0,77831

Tabela 4.1: Entropias do quantizador LBG em um processo de Gauss-Markov

Os processos X_l e Q_j estão relacionados por (4.10). Para um decodificador que obtenha probabilidades *a posteriori* $\beta\left(q_j\right)_{q_j\in I}$ de Q_j, a respectiva estimação de mínima distorção quadrática (MMSE) para os símbolos da fonte associados utiliza o mapa inverso do quantizador φ,

$$\left(\hat{x}_{jm},\ldots,\hat{x}_{jm+m-1}\right) = \frac{\sum_{q_j\in I}\beta\left(q_j\right)\varphi\left(q_j\right).}{\sum_{q_j\in I}\beta\left(q_j\right)} \tag{4.17}$$

Ou seja, a estimação é uma média ponderada dos vetores código $\beta\left(q_j\right)$. Uma média ponderada dos centróides, no caso de um quantizador ótimo.

Para a medida de distorção quadrática, o desempenho do sistema é usualmente avaliado pela relação sinal-ruído entre a sequência X_l original e a reconstituição no destino $\hat{X}_l$ dada por

$$SNR_q = 10 \log_{10} \frac{\sum_l x_l^2}{\sum_l (x_l - \hat{x}_l)^2}. \tag{4.18}$$

4.2.4. O mapeador entre vetores e bits

Nas seções anteriores, foi considerado em linhas gerais a obtenção de um mapa de quantização χ e seus associados ψ e φ com o critério de otimalidade usual, de mínima distorção quando projetado isoladamente. Nesta situação, é irrelevante a indexação.

Posteriormente, quando na presença de erros de transmissão, este codificador pode ter seu desempenho degradado substancialmente. Isso porque a assunção de possíveis adulterações na sequência de vetores código devido a incertezas não foi incluída na otimização.

Em última instância os índices são os elementos que representam a fonte para efeito de transmissão. Quando acoplado a um canal ruidoso, os índices são as variáveis que sofrem possíveis deteriorações, e a preocupação com a indexação passa a ser parte do projeto do sistema.

Para o sistema de transmissão digital, os índices são convertidos em vetores binários, para serem usados em um codificador binário de canal. Neste caso, dado o quantizador, a correspondência entre índices e vetores binários tem influência direta no desempenho final, uma vez que os efeitos do canal são exercidos mais diretamente sobre os símbolos binários, e indiretamente sobre os índices, através deste mapeamento.

Quando o quantizador é dado (projetado separadamente), podemos considerar que um recurso de projeto da codificação atuando sobre este mapeamento faz parte do fundamento codificação de canal. De fato, a codificação de fonte de um sistema típico se resume essencialmente à quantização, isto é, ao mapa χ. Até mesmo os mapas ψ e φ podem ser considerados projetados independentemente, pensando no problema adiante, a preocupação com erros do canal.

O mapeamento entre os índices do quantizador da fonte e vetores binários tem um grau de liberdade de projeto sem custo de complexidade, cuja otimização resulta em sensíveis melhoras de desempenho em algumas situações. Os erros inerentes aos símbolos binários, que sofrem os efeitos do canal mais diretamente, refletem-se nos erros dos índices do quantizador de maneiras diferentes, dependendo de como o mapeamento é realizado. Há uma vasta linha de estudos sobre *index assignment* (IA), vários métodos de otimização e algoritmos. As principais referências são [22], [23], [18], [19], e principalmente [89], onde é apresentado o algoritmo BSA (binary switch algorithm) que obtém mapeamentos localmente ótimos segundo critérios gerais de desempenho. Em nosso sistema e em nossas simulações, consideramos sempre o algoritmo BSA para obtenção de mapeamentos.

Consideramos um mapeador por blocos que a cada vetor Q_j associa um grupo de bits $\{B_{sj}, \ldots, B_{sj+s-1}\}$ respectivamente alinhados nos índices temporais. É qualquer mapa injetor $\Gamma : \{0, \ldots, 2^r - 1\} \rightarrow \{0, 1\}^s$, que associa a cada símbolo r-ário vindo do quantizador uma palavra binária de comprimento s, onde $r \leq s$. Aqui, admitimos a possibilidade de mapeadores que adicionam redundância controladamente, e por isso temos $r \leq s$. Na verdade, isso é equivalente a um mapeador ordinário acoplado a um código de bloco (s, r) acoplado em sua saída.

O projeto do mapeador é feito em conjunto com o quantizador vetorial. Na prática, o que é feito é definir as palavras-código resultantes deste código, e a partir de algoritmos de mínima distorção no quantizador vetorial define-se a associação entre os vetores e estas palavras-código. De acordo com nossas simulações, o algoritmo BSA resulta em mapeamentos eficientes, sem grandes variações de desempenho na distorção média entre dois mapas localmente ótimos quaisquer.

4.2.5. Codificação de canal

O esquema geral permite concatenações turbo de qualquer natureza na codificação de canal para proteção de erros. Aqui, consideramos duas possibilidades. Uma delas é o caso em que o codificador de canal tem embutido um código turbo padrão, com dois geradores

de paridade convolucionais junto à emissão dos bits não codificados, entrelaçamentos, puncionamentos, etc.

Outra possibilidade considera o fato de já haver outros componentes concatenados seriamente que têm memória e redundância, e não há necessidade de uma composição turbo (dois componentes de paridades internos para interagir na decodificação). Basta que o codificador de canal tenha um componente com memória, e que a decodificação iterativa se processe na concatenação serial com os componentes relacionados à fonte (quantizador e mapeamento). Neste caso, temos uma liberdade maior de onde podemos alocar a redundância disponível, ou para codificação de canal ou para proteção dos índices da fonte, através de mapeamento binário com redundância.

Códigos binários convolucionais recursivos, truncados para um dado comprimento de bloco, são usuais para uma decodificação turbo. Anteriormente ao código de treliça, deve haver acoplado um entrelaçador de bloco para quebrar a memória sequencial da redundância dos bits oriundos da quantização-fonte. Analogamente aos sistemas turbo usuais, o entrelaçador define o atraso de codificação e decodificação do sistema como um todo. Em nossas simulações, foram gerados vários entrelaçadores pseudo-aleatórios do tipo *S-random* pelo eficiente algoritmo de Crozier.

Em relação aos códigos empregados, aqui não necessariamente temos códigos sistemáticos. De fato, é comum a utilização de uma codificação convolucional recursiva de taxa 1, ou seja, apenas um gerador de paridade a partir dos bits entrelaçados. Isso não resulta em queda de desempenho, contanto que a taxa extra seja alocada para adicionar redundância no mapeador binário.

Terminando a descrição do sistema empregado, consideramos um canal sem memória, um típico B-AWGN. Um canal com memória poderia ser facilmente inserido neste contexto, e também decodificado conjuntamente.

4.3. Representação por grafos-fatores

Uma vez descritos os componentes, podemos esquematizar o grafo-fator genérico do sistema proposto. Uma primeira versão é dada na figura 4.2.

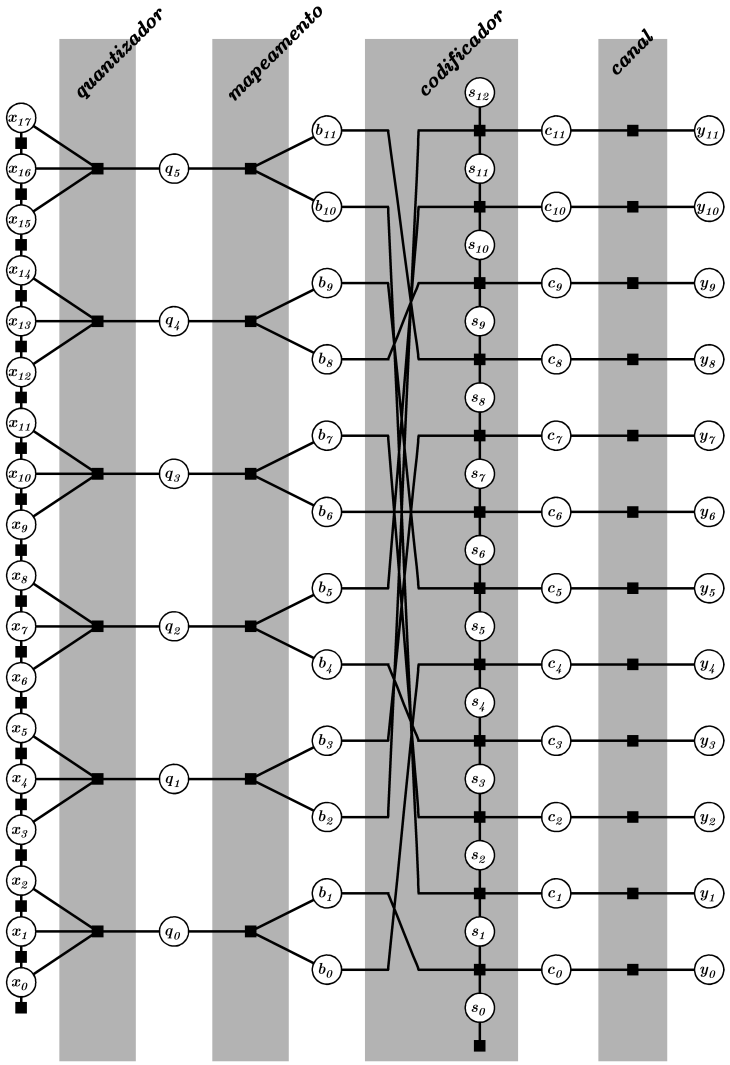

Figura 4.2: Grafo-fator básico do sistema completo para decodificação conjunta, com modelo de quantização vetorial integrada.

Embora com algumas simplificações para viabilizar seu desenho, a generalidade é mantida. Grafos-fatores com outros parâmetros podem ser deduzidos a partir deste. No esquema simples, o comprimento de bloco da codificação de canal é $K = 12$, consideramos uma

quantização vetorial com $m = 3$ dimensões, e uma fonte modelada por um AR de primeira ordem, um processo de Gauss-Markov por exemplo. O mapeamento binário considera 2 bits para cada índice. A figura 4.2 representa portanto um esquema de codificação fonte com $\frac{2}{3}$ bits/símbolo.

Pela esquema da figura 4.2, fica ilustrado que os ciclos no grafo-fator do sistema considerado se estendem desde o codificador de canal até a própria fonte e seus nós do modelo de transição de estados.

4.4. Decodificação sub-ótima MAP

O sistema de decodificação deve reconstruir a sequência X_l dada a sequência de observações Y_t. Considere a decodificação realizada em blocos de comprimento L. Fixado o bloco $\{X_0, \ldots, X_{L-1}\}$, sejam as sequências truncadas correspondentes $\{Q_0, \ldots, Q_{J-1}\}$, $\{B_0, \ldots, B_{K-1}\}$, $\{C_0, \ldots, C_{T-1}\}$ e $\{Y_0, \ldots, Y_{T-1}\}$ internas ao bloco. Por construção, temos $L/K = m/r$ e K/T é igual à taxa do sistema de codificação de canal.

Dadas as variáveis observadas $\{Y_0 = y_0, \ldots, Y_{T-1} = y_{T-1}\}$ e considerando como medida de distorção o erro quadrático médio, a decodificação ótima é a que minimiza o valor esperado da distorção, condicionada à sequência de variáveis observadas. Seja z_l o valor estimado de X_l, $\{z_0, \ldots, z_{L-1}\}$ são tais que

$$D = \min_{\{z_0,\ldots,z_{L-1}\}} E\left[\sum_{l=0}^{L-1} \frac{(X_l - z_l)^2}{L} \,|y_0, \ldots, y_{T-1}\right], \tag{4.19}$$

onde $\{y_0, \ldots, y_{T-1}\}$ são os valores observados.

Portanto, pelo critério de erro quadrático médio mínimo,

$$z_l = E\left[X_l \,|y_0, \ldots, y_{T-1}\right], \; l = 0, \ldots, L-1. \tag{4.20}$$

Antes de desenvolver a expressão (4.20) e chegar ao que queremos, devemos considerar a expressão dos centróides associados ao codificador de fonte dado.

Considerando os primeiros elementos da sequência X_l, o mapa de quantização $\chi : R^m \rightarrow \{0, \ldots, 2^r - 1\}$ faz a associação $(X_0, \ldots, X_{m-1}) \rightarrow Q_0$.

O conjunto de centróides $\{E\left[(X_0, \ldots, X_{m-1})\,|Q_0 = q\right],\ q = 0, \ldots, 2^r - 1\}$ são intrínsecos ao sistema de codificação. Sua expressão é dada por

$$E\left[(X_0, \ldots, X_{m-1})\,|q_0\right] = \int (x_0, \ldots, x_{m-1})\, p\left(x_0, \ldots, x_{m-1}|q_0\right) dx_0 \ldots dx_{m-1}. \tag{4.21}$$

Para cada um destes vetores, cada coordenada pode ser expressa separadamente. Como exemplo, tomando a primeira coordenada correspondente a X_0, vem

$$\begin{aligned} E\left[X_0|q_0\right] &= \int x_0 p\left(x_0, \ldots, x_{m-1}|q_0\right) dx_0 \ldots dx_{m-1} \\ &= \int x_0 p\left(x_0|q_0\right) dx_0. \end{aligned} \tag{4.22}$$

Voltando à expressão da decodificação ótima e fixando a análise à z_0, o primeiro elemento da sequência de símbolos estimados,

$$\begin{aligned} z_0 &= E\left[X_0\,|y_0, \ldots, y_{T-1}\right] \\ &= \int x_0 p\left(x_0|y_0, \ldots, y_{T-1}\right) dx_0. \end{aligned} \tag{4.23}$$

Como vale, aproximadamente,

$$p\left(x_0|y_0, \ldots, y_{T-1}\right) = \sum_{q_0 \in \{0,\ldots,2^r-1\}} p\left(x_0|q_0\right) p\left(q_0|y_0, \ldots, y_{T-1}\right), \tag{4.24}$$

então

$$\begin{aligned} z_0 &= \int x_0 \sum_{q_0 \in \{0,\ldots,2^r-1\}} p\left(x_0|q_0\right) p\left(q_0|y_0, \ldots, y_{T-1}\right) dx_0 \\ &= \sum_{q_0 \in \{0,\ldots,2^r-1\}} \left[\int x_0 p\left(x_0|q_0\right) dx_0\right] p\left(q_0|y_0, \ldots, y_{T-1}\right). \end{aligned} \tag{4.25}$$

O problema central na decodificação portanto reduz-se à obtenção da distribuição de probabilidade $p\left(q_0|y_0, \ldots, y_{T-1}\right)$. Dada esta distribuição, basta usá-la em uma média ponderada dos centróides para obtenção de z_0. Essa dedução vale por analogia para qualquer z_l.

A distribuição de probabilidade $p\left(q_0|y_0,\ldots,y_{T-1}\right)$ vem da decodificação iterativa entre as sequências de variáveis $\{Q_0,\ldots,Q_{J-1}\}$, $\{B_0,\ldots,B_{K-1}\}$, $\{C_0,\ldots,C_{T-1}\}$ e $\{Y_0,\ldots,Y_{T-1}\}$ internas ao bloco. Se o quantizador usado é de bloco e sem memória, obtemos uma sequência Q_j com memória residual. A obtenção da distribuição $P\left(Q_j|Q_{j-1}\right)$ pode ser obtida analiticamente ou por simulações numéricas.

4.5. Decodificação sub-ótima e contínua

Podemos ilustrar um caso da decodificação conjunta iterativa no contexto de grafos-fatores envolvendo todas as variáveis típicas do nosso modelo, inclusive as contínuas. Neste caso teremos nós em que o algoritmo SP envolve integrações em vez de somas. O grafo-fator da figura 4.3 ilustra a transmissão por um sistema digital da fonte dada por um processo AR de primeira ordem, quantização vetorial bidimensional, mapeamento binário, codificador e canal. Em uma decodificação iterativa conjunta, cabe ao decodificador de fonte realizar uma inferência sequencial do processo aleatório (*forward-backward* com variáveis contínuas) e trocar informação probabilística símbolo a símbolo com o decodificador de canal.

Podemos descrever resumidamente o conjunto de funções envolvidas em cada nó da fatoração pelo grafo da distribuição conjunta.

O esquema ilustra as variáveis X_i evoluindo de acordo com um processo AR de primeira ordem, um processo de Gauss-Markov por exemplo. A quantização associa a cada par de variáveis (X_i, X_{i+1}) a variável discreta $Q_i \in \{0,\ldots,2^r-1\}$. Na figura 4.3, temos o caso particular $r = 3$ ilustrado. Desta forma, fica representado no mesmo nó duas funções: a correlação entre variáveis X_{i+1} e X_i

$$Pr\left\{x_{i+1}|x_i\right\} \triangleq Pr\left\{X_{i+1}=x_{i+1}|X_i=x_i\right\}, \tag{4.26}$$

e o mapa da quantização bidimensional

$$Pr\left\{q_i|x_i,x_{i+1}\right\} \triangleq Pr\left\{Q_i=q_i|X_i=x_i,X_{i+1}=x_{i+1}\right\}. \tag{4.27}$$

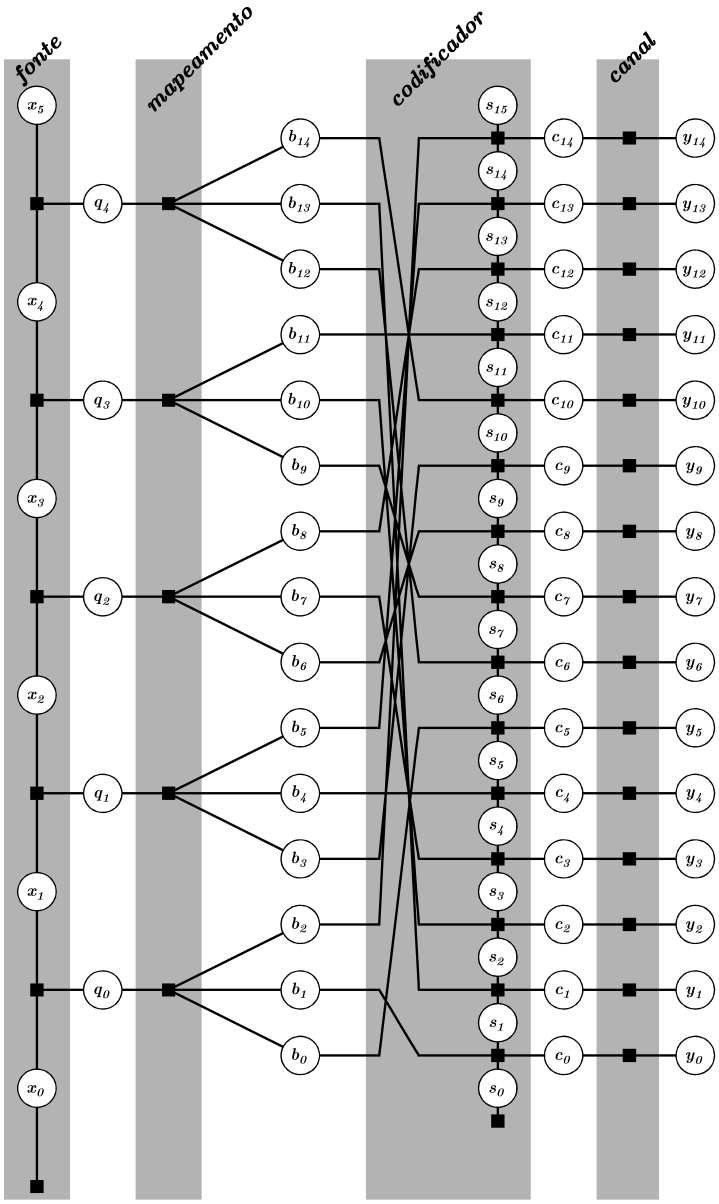

Figura 4.3: Grafo-fator num caso particular, para ilustrar a decodificação incluindo variáveis contínuas da fonte.

A função resultante $T(x_i, x_{i+1}, q_i)$ no nó do grafo é o produto

$$f_F(x_i, x_{i+1}, q_i) = Pr\{x_{i+1}|x_i\} Pr\{q_i|x_i, x_{i+1}\} \tag{4.28}$$

No mapeamento, cada Q_i gera as variáveis binárias $\{B_{ri}, \ldots, B_{ri+r-1}\}$. A função no nó

envolve apenas variáveis discretas, dada por

$$f_M\left(q_i, b_{ri}, \ldots, b_{ri+r-1}\right) = Pr\left\{b_{ri}, \ldots, b_{ri+r-1} | q_i\right\}. \tag{4.29}$$

Para o entrelaçador, o codificador e o canal as funções envolvidas são as usuais,

$$f_E\left(b_j, s_k, s_{k+1}, c_k\right) = Pr\left\{s_{k+1} | b_j, s_k\right\} Pr\left\{c_k | b_j, s_k\right\} \tag{4.30}$$

e

$$f_C\left(c_k, y_k\right) = Pr\left\{y_k | c_k\right\} \tag{4.31}$$

Daqui ficam definidas as equações do algoritmo SP aplicado ao esquema. Particularmente, no bloco fonte/quantização, as equações são

$$\alpha\left(x_{i+1}\right) = \sum_{q_i=0}^{2^r-1} \int_{x_i=-\infty}^{+\infty} f_F\left(x_i, x_{i+1}, q_i\right) \beta\left(q_i\right) \alpha\left(x_i\right), \quad -\infty < x_{i+1} < +\infty; \tag{4.32}$$

$$\beta\left(x_i\right) = \sum_{q_i=0}^{2^r-1} \int_{x_{i+1}=-\infty}^{+\infty} f_F\left(x_i, x_{i+1}, q_i\right) \beta\left(q_i\right) \beta\left(x_{i+1}\right), \quad -\infty < x_i < +\infty; \tag{4.33}$$

$$\alpha\left(q_i\right) = \int_{x_i=-\infty}^{+\infty} \int_{x_{i+1}=-\infty}^{+\infty} f_F\left(x_i, x_{i+1}, q_i\right) \alpha\left(x_i\right) \beta\left(x_{i+1}\right), \quad q_i = \{0, \ldots, 2^r - 1\}. \tag{4.34}$$

Terminada a decodificação iterativa, resulta a densidade marginal $\alpha\left(x_i\right)\beta\left(x_i\right)$ relativa à variável X_i, e a decisão de mínima distorção quadrática é a decisão pela média desta densidade,

$$z_i = \int_{x_i=-\infty}^{+\infty} \alpha\left(x_i\right)\beta\left(x_i\right). \tag{4.35}$$

Caberia assim ao decodificador de fonte realizar uma estimação sequencial do processo aleatório (*forward-backward* com variáveis contínuas) e trocar informação probabilística símbolo a símbolo com o decodificador de canal, através do mapa de quantização.

O exposto acima ilustra a complexidade envolvida para implementar a decodificação iterativa conjunta em sua plenitude. No caso acima, com quantização vetorial bidimensional, temos integrais duplas. A complexidade aumenta à medida que queremos uma quantização

vetorial em mais dimensões. A decodificação completa de um esquema como o da figura 4.2 envolveria integrais multidimensionais. O método natural de implementação (integração numérica) resulta numa complexidade proibitiva em se tratando de um algoritmo iterativo.

4.6. Redução ao caso discreto

O exposto na seção anterior descreve uma decodificação que considera todas as variáveis na distribuição conjunta associada à fatoração, inclusive as variáveis contínuas X_l. Como visto na seção 4.4, podemos reduzir a análise à obtenção das distribuições *a posteriori*

$$p\left(q_j | y_0, \ldots, y_{T-1}\right). \tag{4.36}$$

Ou seja, o problema geral de decodificação conjunta entre variáveis discretas e contínuas pode ter sua complexidade reduzida, onde a única hipótese é que vale a aproximação dada na fórmula (4.24).

No contexto de grafos-fatores, isso é equivalente a dizer que podemos simplificar a distribuição conjunta utilizada. Em vez de

$$p\left(x_0, \ldots, x_{L-1}, q_0, \ldots, q_{J-1}, b_0, \ldots, b_{K-1}, c_0, \ldots, c_{T-1}, y_0, \ldots, y_{T-1}\right), \tag{4.37}$$

a decodificação reduz-se a obter marginais a partir de

$$p\left(q_0, \ldots, q_{J-1}, b_0, \ldots, b_{K-1}, c_0, \ldots, c_{T-1}, y_0, \ldots, y_{T-1}\right). \tag{4.38}$$

O esquema inicial proposto na figura 4.2 pode ser traduzido em um novo esquema (equivalente ou aproximadamente equivalente), representado na figura 4.4, onde a redundância residual é modelada nas variáveis discretas (no caso dos esquemas acima, as variáveis Q_k).

Heuristicamente, estamos considerando que a estatística (modelo) do processo contínuo X_k induz a estatística do processo discreto Q_k de forma bem definida (função do mapa de quantização). Seria possível, em princípio por cálculo analítico, derivar o comportamento do processo Q_k a partir de X_k. Mas mesmo quando isso não é realizável, pode-se extrair

o comportamento de Q_k estatisticamente, por sequências de treinamento. A redundância residual fica assim implícita no modelo do processo Q_k, e apenas este modelo será envolvido no algoritmo de decodificação iterativa. De fato, devido ao processo de codificação digital de fonte (redução da taxa de entropia), somente a redundância residual pode auxiliar na decodificação conjunta.

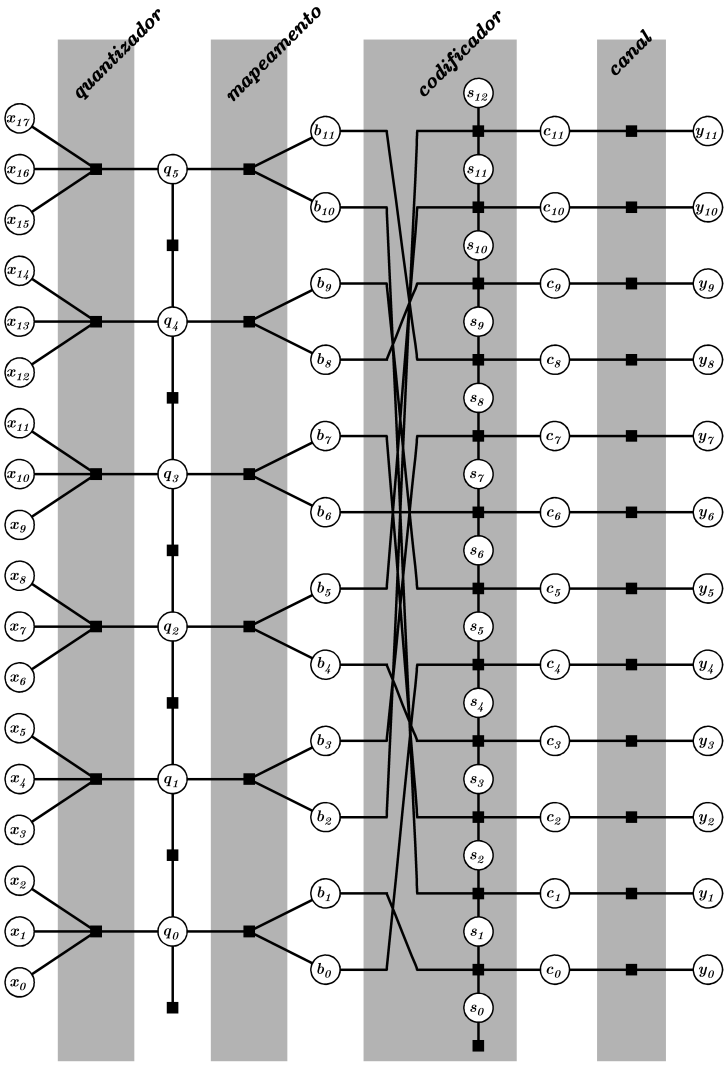

Figura 4.4: Nova versão para o grafo-fator do sistema proposto, com a aproximação e redução ao caso da modelagem por $p\left(q_{j+1}|q_j\right)$.

A decodificação conjunta iterativa é então realizada apenas utilizando informação do processo Q_k, envolvendo um *forward-backward* nesta sequência de variáveis. Após a saturação

do algoritmo iterativo, temos como resultado probabilidades *a posteriori* $p\left(q_0|y_0,\ldots,y_{T-1}\right)$. A decisão nas variáveis X_k é obtida por critério de erro quadrático médio mínimo, através da equação (4.25).

4.7. Simulações

A partir do esquema ilustrado 4.4 acima, curvas de desempenho por simulações podem então ser obtidas envolvendo codificadores de fonte com quantização vetorial, seja isoladamente ou conjunta com o decodificador de canal. Destacamos aqui alguns resultados obtidos por cálculos computacionais simulando alguns sistemas com parâmetros típicos. São curvas de SNR_q de reconstituição da fonte *versus* SNR de uso de canal. Ou seja, relações sinal/ruído, ambas em dB. A equação (4.18) define a SNR_q de reconstituição da fonte, uma medida baseada na distorção quadrática média resultante.

Um primeiro estudo relevante são simulações comparando as curvas de desempenho entre duas situações:

- decodificação conjunta iterativa entre decodificador de fonte e decodificador de canal,
- decodificação iterativa apenas no decodificador de canal.

A figura 4.5 apresenta curvas de desempenho obtidas por simulações do algoritmo SP no grafo-fator da concatenação nas duas situações. A fonte é Gauss-Markov com coeficiente $\alpha = 0{,}9$. O quantizador vetorial codifica com 4 bits/4 amostras, e portanto o bloco usa 4096 amostras da fonte. O codificador de canal é um turbo (concatenação paralela com entrelaçador) de taxa 1/3 de 16 estados com geradores de paridade $\left(1+D^3+D^4\right)/\left(1+D+D^2+D^4\right)$ e comprimento de bloco 16384 bits. As curvas mostram os desempenhos para 3, 6, 9, 12, e 15 iterações nos dois casos.

As curvas já estão transladadas no eixo de SNR de uso de canal considerando a entropia H do processo Q_j como redundância intrínseca, calculada a partir das probabilidades de transições $P\left(Q_{j+1}|Q_j\right)$ obtidas por sequências de treinamento suficientemente extensas.

Aqui, com $\alpha = 0{,}9$, $m/r = 4/4$, e um quantizador LBG, a entropia binária residual vale $H\left(Q_{j+1}|Q_j\right) = 3{,}13202$. O ajuste é da forma

$$SNR = SNR^* - 10log_{10}\left(H\left(Q_{j+1}|Q_j\right)/r\right), \tag{4.39}$$

onde SNR^* é o valor considerando apenas o codificador de canal, o que é equivalente ao usual E_b/N_0 para seus bits de entrada B_k.

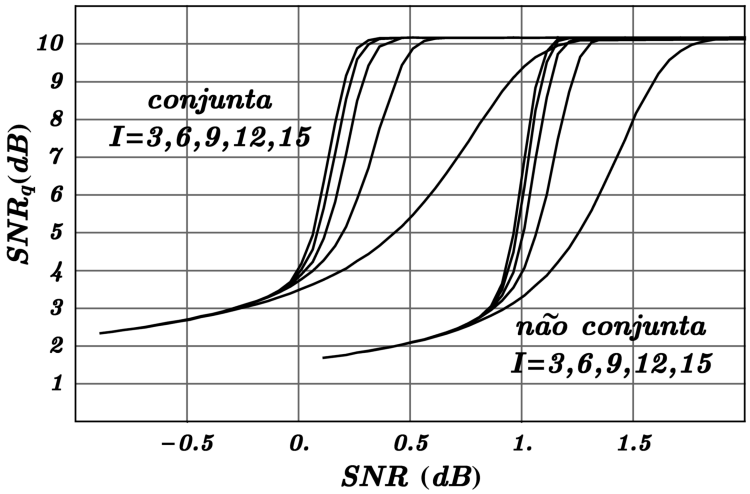

Figura 4.5: Curvas de reconstituição da fonte para decodificação conjunta e não conjunta, com diferentes números de iterações.

Na comparação entre as curvas, vemos que a integração da informação da fonte na decodificação iterativa resulta em ganhos significativos. Em $SNR = 0{,}3dB$, a região de maiores discrepâncias, obtemos um ganho pela decodificação combinada da ordem de $0{,}8dB$ em SNR_q.

Também são interessantes análises de ganho entre diferentes complexidades do codificador de fonte. Para um mesmo codificador de canal e fixada a taxa do codificador de fonte, a figura 4.6 ilustra os resultados de desempenho para $m/r = 3/3$, $4/4$, e $5/5$. As entropias binárias resultantes dos três quantizadores vetoriais são respectivamente $H\left(Q_{j+1}|Q_j\right) =$ 2,51869, 3,50569, e 4,48972. O processo X_l é um Gauss-Markov de coeficiente $\alpha = 0{,}8$. O codificador de canal acoplado é um codificador turbo com geradores de paridade $\left(1 + D^3 + D^4\right) / \left(1 + D + D^2 + D^4\right)$ e comprimento de bloco 16380.

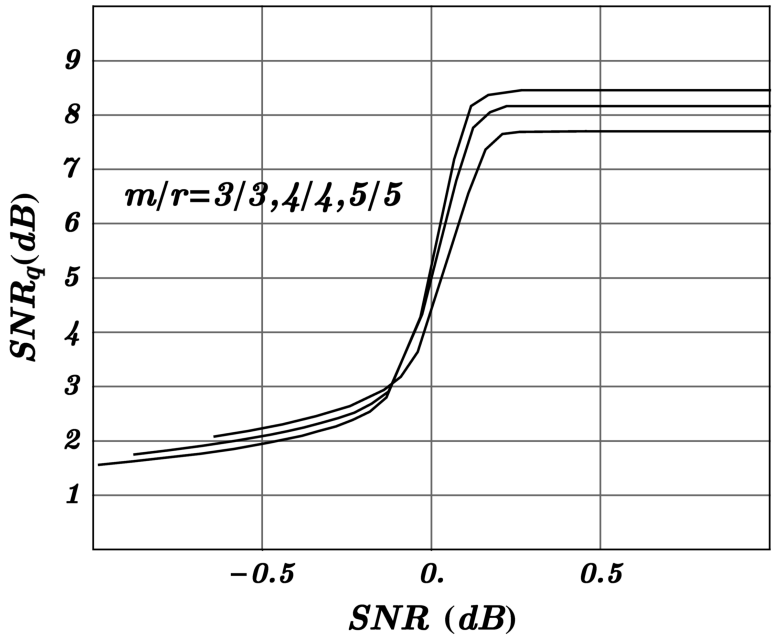

Figura 4.6: Curvas de reconstituição da fonte para diferentes codificadores de fonte, taxa 1bit/símbolo fixa.

Aqui, diferentes taxas implicam em considerável aumento na complexidade envolvida na decodificação. Quanto maior r, mais estados no decodificador.

As simulações mais interpretativas são os casos em que o processo X_k é Gauss-Markov (auto-regressivo de primeira ordem). Entretanto, o método de aproximação vale também para fontes com processos auto-regressivos e/ou médias-móveis (ARMA) quaisquer. Ainda que não seja possível obter analíticamente de forma exata a estatística que o processo contínuo X_k induz no processo discreto Q_k, é sempre possível obter aproximadamente por simulação das sequências. Como ilustração, a seguir apresentamos os resultados de simulação para um processo ARMA estacionário dado por $X_{l+1} = W_l + 0{,}41X_l + 0{,}27X_{l-1} + 0{,}17X_{l-2}$.

O codificador de canal é um codificador turbo de taxa 1/2 (puncionado) e 16 estados com geradores de paridade $(1 + D^4) / (1 + D + D^2 + D^3 + D^4)$ e comprimento de bloco 16380 bits. A figura 4.7 apresenta os resultados de desempenho para $m/r = 3/3$ e $4/4$. As entropias binárias de transição para os símbolos emitidos pelo quantizador são $H(Q_{j+1}|Q_j) = 2{,}35269$ e 3,37614, respectivamente.

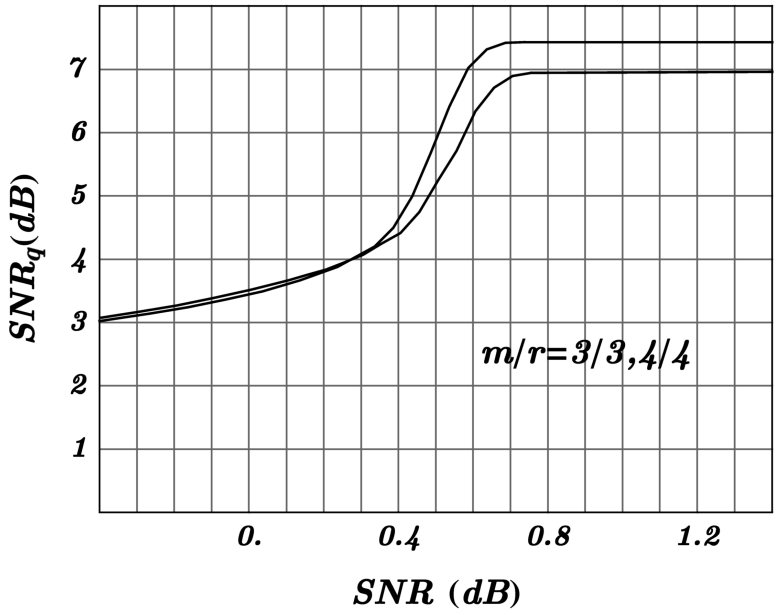

Figura 4.7: Curvas de reconstituição da fonte para diferentes codificadores de fonte modelada como um processo AR.

Os exemplos evidenciam as vantagens da decodificação iterativa conjunta. Além disso, nas simulações percebe-se que não há necessidade do recurso da proteção desigual de erros nos bits emitidos pelo quantizador. Isso deve-se à inserção da informação de redundância do processo Q_j na decodificação.

Já na figura 4.8 apresentamos curvas do sistema utilizando mapeamentos binários com redundância (RIA), como em [14]. Neste tipo de sistema, o mapeamento consiste em um componente de taxa 1/2, deixando de usar um codificador de canal turbo padrão. Aqui, o codificador tem taxa 1, sendo um gerador de paridade convolucional recursivo dado por $1/\left(1 + D + D^2 + D^3\right)$, como propriamente ilustrado nas figuras 4.2 e 4.4 anteriores.

A análise em [14] indica que quando podemos escolher onde a redundância adicional disponível (de taxa 1/2) pode ser alocada, o melhor é justamente optar pelo componente intermediário entre quantizador e codificador de canal, isto é, no mapeamento binário. Como em [14], o melhor tipo de mapa revelou ser (por simulações entre vários) o mapa de repetição, otimizado pelo algoritmo BSA. Podemos assim comparar curvas com fontes de Gauss-Markov

de variados parâmetros α e quantizadores vetoriais LBG de variadas dimensões, mantida a mesma taxa $m/r = 3/3$, $4/4$, $5/5$. O comprimento de bloco é fixo em 16380.

As três curvas com $m/r = 3/3$ e diferentes parâmetros $\alpha = 0{,}80$, $0{,}90$ e $0{,}95$ ilustram a gradativa melhora da decodificação à medida que consideramos maior determinismo no processo na fonte. Para uma comparação justa, todas as curvas estão ajustadas de acordo com as taxas de entropias dos processos Q_j na saída dos quantizadores, de acordo com a expressão (4.39).

Para outro tipo de análise, as três curvas com mesmo parâmetro $\alpha = 0{,}95$ e variadas dimensões $m = 3, 4, 5$ indicam o gradativo ganho em resolução. Aqui se observa que entre $m = 4$ e 5 o ganho em resolução não resulta em perdas de região de não-deterioração, indicando um caso em que a decodificação iterativa conjunta cumpriu bem seu papel.

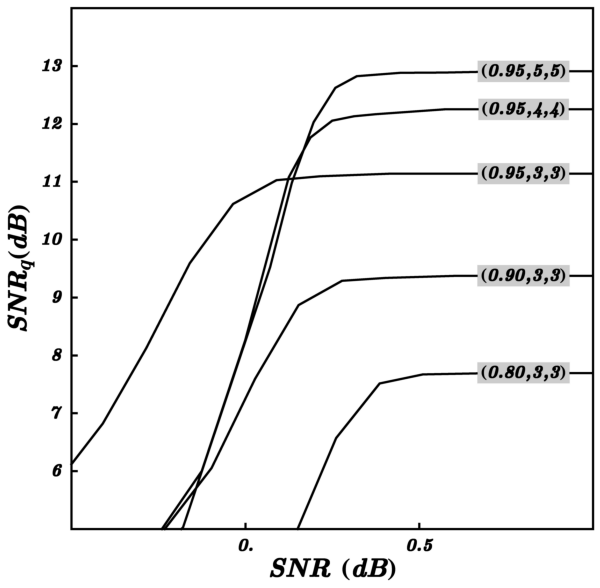

Figura 4.8: Curvas de reconstituição da fonte para diferentes parâmetros (α, m, r), com $m/r = 1$.

4.8. O esquema de decodificação de Goertz

Um dos objetivos postulados em nosso projeto era observar a literatura existente sobre decodificação iterativa combinada fonte-canal e inserir seus modelos no contexto de grafos-fatores. Na literatura científica consagrada sobre o tema ISCD, uma linha de abordagem comum é dada por Goertz, Hindelang e outros [41][74][48][36]. O texto de Goertz [36] é a referência principal e definitiva para ISCD, a partir do qual todos os desenvolvimentos seguintes foram baseados. Nesta linha, é formulada uma maneira de se obter as informações extrínsecas da decodificação turbo dentro do contexto conjunto e como elas são trocadas entre blocos de decodificação. As extrínsecas do decodificador de fonte estão diretamente relacionadas à redundância residual.

Podemos dizer que esta linha de desenvolvimento não implementa de fato uma decodificação iterativa completa, uma vez que a preocupação com o atraso de decodificação leva a um esquema que não explora completamente a redundância residual do codificador de fonte. O esquema de Goertz considera decodificação em blocos envolvendo apenas um índice de quantização, e aproveita apenas a informação *a priori* dos índices anteriores. Assim, há passagem de informação do decodificador de fonte para auxiliar o decodificador de canal, mas sem de fato realizar uma decodificação conjunta e iterativa entre as variáveis dos dois blocos.

É possível sintetizar os esquemas desta literatura em apenas um grafo-fator, representado na figura 4.9. Essencialmente, estes esquemas utilizam um modelo de Markov de primeira ordem para os símbolos quantizados emitidos pelo codificador de fonte, e somente quantização escalar é considerada. A figura 4.9 ilustra cada elemento da concatenação quando representados em um grafo-fator: uma fonte markoviana de símbolos q-ários $\{Q_k\}_{k=0,\ldots}$ (aqui representados apenas os símbolos iniciais da sequência de variáveis, dispostos na vertical), um mapeamento genérico, e um codificador de canal (entrelaçador e código de treliça).

Para o cronograma para o grafo-fator da figura 4.9, a passagem de mensagens é restrita a um bloco por vez. E cada bloco de decodificação contém apenas um símbolo Q_k. O entrelaçamento também é interno a cada bloco.

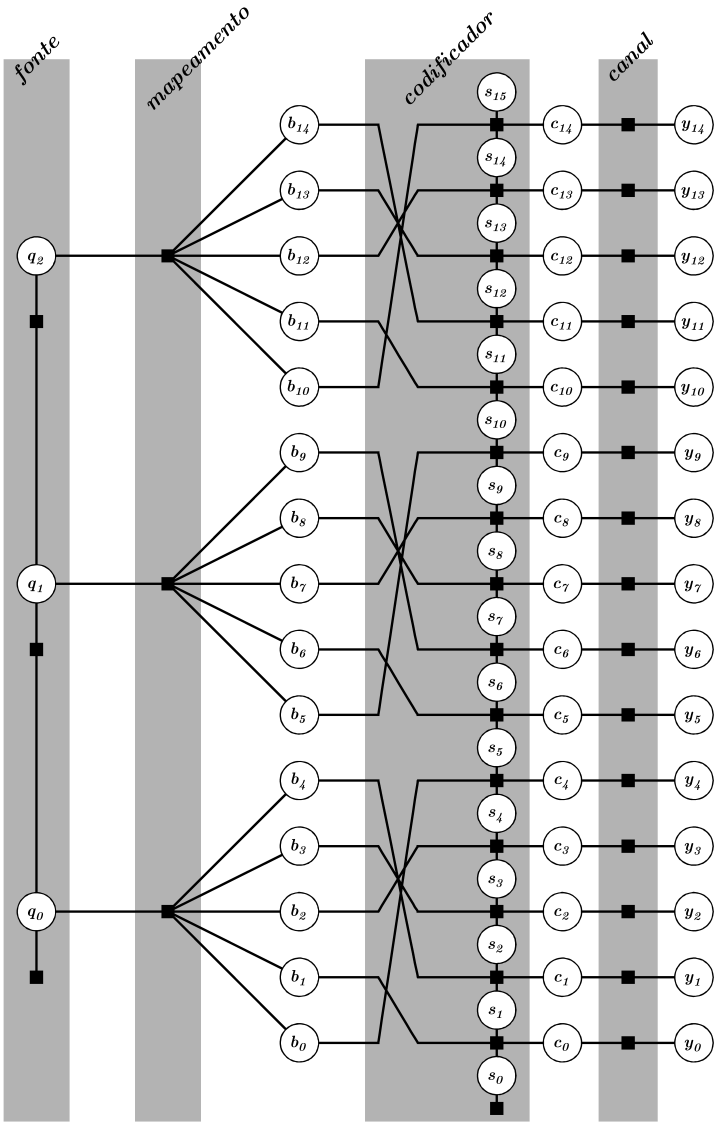

Figura 4.9: Tradução para grafos-fatores do esquema de decodificação ISCD de Gortz.

Baseados em esquemas deste tipo (mas fora do contexto de grafos-fatores), a maior parte dos artigos da literatura sobre o tema implementa uma decodificação supostamente conjunta onde o decodificador de canal aproveita a informação *a priori* dos símbolos q-ários da fonte para melhorar sua inferência sobre os bits no codificador de canal. Essa informação *a priori* é obtida com a realização da parte *forward* do algoritmo *forward-backward* na fonte markoviana. Percebe-se, portanto, que a aplicação da decodificação iterativa quando há forte restrição de atraso, como é o modelo desenvolvido na linha de pesquisa de Goertz,

não aproveita plenamente a memória e redundância residual na saída do codificador de fonte.

4.9. Considerações finais sobre o capítulo

O estudo da decodificação combinada fonte-canal completa utilizando o ambiente de grafos-fatores e a obtenção de seu desempenho é ainda inexplorado na literatura. Nosso sistema considera quantização vetorial e processos ARMA na modelagem da fonte, que estabelece um nível de maior generalidade em relação aos modelos de decodificação combinada existentes na literatura atual. A análise da decodificação no contexto de grafos-fatores representa sobretudo um avanço no entendimento da otimização de tais sistemas, que envolvem variáveis contínuas e estimações. Os resultados das simulações computacionais são originais e confirmam a relevância da modelagem obtida.

Em geral, em sistemas envolvendo várias variáveis contínuas, é proibitivo integrá-las a uma estimação/decodificação iterativa completa, devido à dificuldade de contornar a complexidade do algoritmo (versão integral-produto do SP). Uma linha de pesquisa representada por [88][87][17][67][76] com esquemas de grafos-fatores envolvendo modelos de canais de Rayleigh (desvanecimento e/ou não coerente) têm utilizado alguns métodos de aproximação para a computação do algoritmo SP nas variáveis contínuas envolvidas: ou por quantização das mensagens [17], ou pelas chamadas distribuições canônicas [88][87]. Outros autores fazem uma adaptação de filtros de Kalman para realizar uma aproximação do que seria a inferência sequencial de um processo aleatório AR [67][76]. Isso demanda assumir ad-hoc que a aproximação gaussiana é boa para as densidades de probabilidade das variáveis do processo aleatório.

Em [10] uma visão unificada de diversos métodos de decodificação e detecção multiusuário conjunta foi elaborada a partir de grafos-fatores. Em [80] uma visão unificada de diversos métodos de decodificação e equalização conjunta foi também elaborada a partir de grafos-fatores. O trabalho apresentado neste capítulo oferece uma visão unificada para um método de decodificação conjunta fonte e canal na linha de contornar a complexidade para envolver variáveis contínuas em uma estimação iterativa pelo algoritmo SP.

Capítulo 5

Conclusões e futuros trabalhos

Essencialmente, esta tese analisou aplicações relevantes da teoria de grafos-fatores e seu algoritmo SP. O capítulo 2 apresentou uma compilação diferenciada sobre a teoria de grafos-fatores, com uma abordagem de pontos e interpretações relevantes, inéditas na literatura. O capítulo 3 contém a construção do esquema de decodificação turbo imerso na teoria de grafos-fatores, de uma forma apropriada e genérica, que permitiu analisar o esquema de decodificação turbo *stream-oriented* de forma inovadora. No capítulo 4, apresentamos um estudo sobre como modelar o problema geral de transmissão de uma fonte contínua através de um canal ruidoso utilizando estas ferramentas conhecidas para a decodificação, incluindo resultados selecionados de simulações. Assim, cada um destes três capítulos contém material relevante e inédito. Além disto, as estrutuuras mais gerais contruídas aqui permitem colocar vários esquemas de codificação já existentes no mesmo contexto, proporcionando uma visão geral sobre estes, além de criar as condições para novos esquemas e cronogramas, obtendo novas variantes de algoritmos iterativos.

No capítulo 3, temos uma situação em que a relevância está na descrição precisa do cronograma de decodificação, e no capítulo 4 temos uma situação em que temos maior arbitragem sobre a modelagem dos componentes, sobre a realização do grafo-fator, além do cronograma de decodificação. Ao longo destes desenvolvimentos, ficou evidente que as várias possibilidades de projetos possíveis são melhor visualizadas e entendidas devido à presença

da teoria de grafos-fatores.

O tema de grafos-fatores é atual e unifica muitas idéias em teoria de informação, teoria de codificação e teoria de comunicação. Pode ser considerado um genuíno tópico de interseção entre todas elas. É relacionado à codificação de canal, equalização, modulação, além de envolver tópicos em processos estocásticos, inferência, etc. O tema ainda se encontra em pleno desenvolvimento, com novas idéias e aplicações surgindo a todo momento.

Entendemos que a forma de representação original para esquemas de grafos-fatores (inspirada nos grafos de Tanner, mas generalizada) ainda é a melhor (em relação às suas equivalentes, como grafos normais de Forney), por manter uma heurística entre variáveis e funções como elementos do esquema. Além disso, a partir de uma representação usual, a representação de Forney, à qual foi atribuída a nomenclatura FFG (*Forney Factor-Graphs*), pode ser considerada equivalente, mas é menos construtiva e quase sempre se deriva da usual por transformações simples. Entretanto, é notável que a introdução criteriosa de variáveis clones e nós de igualdade em grafos-fatores frequentemente levam a interessantes entendimentos sobre o sistema descrito em questão.

Dado que a representação de sistemas por grafos-fatores é o resultado de uma série evolutiva de generalizações de modelos gráficos anteriores a ela, uma pergunta relevante que poderíamos formular seria: é possível uma nova representação gráfica mais genérica do que grafos-fatores? No nosso entendimento, não parece provável. Pelo seu caráter fundamental, baseado nos elementos mais básicos de um sistema, variáveis e vínculos, não há espaço para ir além disso. O algoritmo SP, entendido na sua forma mais geral, operando sobre duas operações algébricas quaisquer relacionadas pela distributividade, contempla várias classes de algoritmos conhecidos. Como distributividade é também operação essencial na álgebra abstrata, resta pouco espaço para generalizações. Por outro lado, é consenso que novos esquemas de algoritmos práticos podem ser derivados das inúmeras variações que o algoritmo SP genérico possibilita, como casos particulares.

Como sugestões de trabalhos futuros, podemos citar uma melhor caracterização do fluxo de informação no cronograma dado para decodificação *stream-oriented*, e algumas variações com atraso de decodificação reduzidas, com perda de desempenho aceitável. Como

sugerido no texto do capítulo 3, ainda foi pouco explorada a problemática da decodificação turbo contínua, relevante para projetos de sistemas práticos com preocupações de atraso, causalidade e sincronismo. O esquema de decodificação *stream-oriented* foi apresentado para o caso básico da concatenação paralela, mas pode ser facilmente modificado por analogia para concatenações seriais com o mesmo princípio.

Uma análise comparativa entre sistemas turbo *stream-oriented* e sistemas turbo de bloco em pequenos comprimentos é possível, e já foi parcialmente concluída na literatura existente. Entretanto, esta análise tende a desconsiderar o efeito de atraso intrínseco ao turbo *stream-oriented* em função do número de iterações, o que não ocorre em sistemas turbo de bloco. A causalidade, natural ao esquema de codificação *stream-oriented*, resulta neste efeito (indesejado em alguns casos). Assim, uma comparação genérica não faria sentido, devendo ser específica e considerando os parâmetros particulares do caso prático em questão.

Modelos gráficos esparsos e algoritmos baseados em propagação de mensagens são objetos de estudo da atualidade e ganham cada vez mais notoriedade pela sua generalidade e possível extensão a diversos campos de conhecimento. Particularmente, o tema "códigos sobre grafos" continua sendo uma área de intensa pesquisa.

Em relação à decodificação conjunta iterativa fonte/canal, seria de interesse obter uma melhor caracterização das mensagens, principalmente as q-árias, que fluem através do grafo típico e sua possível relação com os parâmetros do codificador de fonte. A finalidade seria obter algum tipo de análise de convergência e um projeto de codificação/decodificação integrado com melhoras no desempenho resultante.

A quantização vetorial e seus correspondentes mapeamentos entre vetores e índices podem ser pensados como uma nova e inexplorada maneira de lidar com variáveis contínuas no algoritmo SP. De forma geral, qualquer quantização pode ser entendida, no ambiente de grafos-fatores, como uma inserção de conjuntos de variáveis discretas acopladas a conjuntos de variáveis contínuas, com os respectivos nós vinculadores relacionados aos mapas de quantização. Quando esta inserção resulta na transposição do algoritmo SP envolvendo as variáveis contínuas em um novo cronograma aproximadamente equivalente envolvendo apenas as variáveis discretas inseridas, se estabelece uma significante simplificação. Assim,

com critérios de mínima distorção no projeto destes mapas, a partir de uma inferência aproximada nas variáveis quantizadas naturalmente se deriva a inferência nas variáveis contínuas. Esse princípio básico e todas as suas implicações ainda foram pouco explorados na literatura. O que fizemos aqui foi mostrar uma das inúmeras possíveis instâncias em que esta tranposição ocorre.

Por fim, uma investigação sobre a decodificação conjunta contínua completa, com uma implementação do algoritmo SP contínuo com uma complexidade factível ainda pode ser pretendida e estudada. Ainda assim, a ferramenta da redução ao caso discreto aqui apresentada é sempre um recurso menos dispendioso e um guia para indicar os casos (parâmetros de sistema) em que é compensador ir além e procurar esta decodificação mais complexa.

Referências Bibliográficas

[1] M. Adrat & P. Vary & T. Clevorn, *Optimized bit rate allocation for iterative source-channel decoding and its extension towards multi-mode transmission*, Proceedings of IST Mobile and Wireless Communications, pp. 1153–1157, June 2005.

[2] S. M. Aji & R. J. McEliece, *A general algorithm for distributing information on a graph*, Proceedings of the IEEE International Symposium on Information Theory, pp. 6, July 1997.

[3] S. Aji & R. J. McEliece, *The generalized distributive law*, IEEE Transactions on Information Theory, vol. 46, no. 2, pp.325-343, March 2000.

[4] L. R. Bahl & J. Cocke & F. Jelinek & J. Raviv, *Optimal decoding of linear codes for minimizing symbol error rate*, IEEE Transactions on Information Theory, vol. 20, pp. 284-287, March 1974.

[5] S. Benedetto & G. Montorsi, *Unveiling turbo-codes: some results on parallel concatenated coding schemes*, IEEE Transactions on Information Theory, vol. 42, no. 2, pp. 409-429, March 1996.

[6] S. Benedetto & D. Divsalar & G. Montorsi & F. Pollara, *Serial Concatenation of Interleaved Codes: Performance Analysis, Design, and Iterative Decoding*, TDA Progress Report, vol. 42-126, August 1996.

[7] S. Benedetto & G. Montorsi, *Performance of Continuous and Blockwise Decoded Turbo Codes*, IEEE Communications Letters, vol. 1, pp. 77-79, May 1997.

[8] S. Benedetto & D. Divsalar & G. Montorsi & F. Pollara, *Soft-Input Soft-Output Modules for the Construction and Distributed Iterative Decoding of Code Networks*, European Transactions on Telecommunications, vol. 9, no. 2, pp.155-172, March-April 1998.

[9] C. Berrou & A. Glavieux & P. Thitimajshima, *Near Shannon limit error correcting coding: Turbo codes*, Proceedings of IEEE International Conference on Communications, pp. 1064-1070, May 1993.

[10] J. Boutros & G. Caire, *Iterative multiuser decoding: unified framework and asymptotic performance analysis*, IEEE Transactions on Information Theory, vol. 48, no. 7, pp. 1772-1793, July 2002.

[11] K. Chugg & A. Anastasopoulos & X. Chen, *Iterative Detection*, Kluwer Academic Publishers, Boston, 2001.

[12] G. Clark & J. Cain, *Error-Correction Coding for Digital Communications*, Plenum Press, New York, 1981.

[13] T. Clevorn & P. Vary & M. Adrat, *Iterative source-channel decoding using short block codes*, Proceedings of IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP), vol. 4, pp. 221-224, May 2006.

[14] T. Clevorn & L. Schmalen & P. Vary & M. Adrat, *On Redundant Index Assignments for Iterative Source-Channel Decoding*, IEEE Communications Letters, vol. 12, no. 7, July 2008.

[15] T. M. Cover & J. A. Thomas, *Elements of Information Theory*, Wiley, New York, 1991.

[16] S. Crozier, *New High-Spread High-Distance Interleavers for Turbo-Codes*, Proceedings of the 20th Biennial Symposium on Communications, pp. 3-7, May 2000.

[17] J. Dauwels & H. A. Loeliger, *Joint Decoding and Phase Estimation: An Exercise in Factor Graphs*, Proceedings of the IEEE International Symposium on Information Theory, pp. 231, June 2003.

[18] J. R. B. De Marca & N. S. Jayant, *An algorithm for assigning binary indices to the codevectors of a multi-dimensional quantizer*, Proceedings of IEEE International Conference on Communications, pp. 1128-1132, June 1987.

[19] J. R. B. De Marca & N. Farvardin & Y. Shoham, *Robust vector quantization for noisy channels*, Proceedings of the Mobile Satellite Conference, p. 515-520, May 1988.

[20] S. Dolinar & D. Divsalar & F. Pollara, *Code Performance as a Function of Block Size*, TMO Progress Report 42-133, May 1998.

[21] P. Elias, *Coding for noisy channels*, Institute of Radio Engineers Convention Record, vol. 4, pp. 37-46, September 1955.

[22] N. Farvardin & V. Vaishampayan, *Optimal quantizer design for noisy channels: An approach to combined source-channel coding*, IEEE Transactions on Information Theory, vol. IT-33, pp. 827-838, November 1987.

[23] N. Farvardin, *A study of vector quantization for noisy channels*, IEEE Transactions on Information Theory, vol. 36, pp. 799-809, July 1990.

[24] N. Farvardin & V. Vaishampayan, *On the performance and complexity of channel optimized vector quantizers*, IEEE Transactions on Information Theory, vol. 37, pp. 155-160, Jan. 1991.

[25] T. Fingscheidt & P. Vary, *Robust speech decoding: A universal approach to bit error concealment*, Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP), vol. 3, pp. 1667-1670, April 1997.

[26] T. Fingscheidt & P. Vary, *Softbit speech decoding: A new approach to error concealment*, IEEE Transactions on Speech and Audio Processing, vol. 9, pp. 240-251, March 2001.

[27] T. Fingscheidt & T. Hindelang & R. V. Cox & N. Seshadri, *Joint source-channel (de-)coding for mobile communications*, IEEE Transactions on Communications, vol. 50, no. 2, pp. 200-212, February 2002.

[28] G. D. Forney Jr., *Burst-correcting codes for the classic bursty channel*, IEEE Transactions on Communication Technology, vol. 19, pp. 772-781, October 1971.

[29] G. D. Forney, Jr., *Codes on graphs: Normal realizations*, IEEE Transactions on Information Theory, vol. 47, no. 2, pp. 520-548, February 2001.

[30] B. J. Frey, *Graphical Models for machine Learning and Digital Communication*, The MIT Press, 1998.

[31] R. G. Gallager, *Low Density Parity Check Codes*, Sc.D. thesis, Massachusetts Institute of Technology (MIT), September 1960.

[32] R. G. Gallager, *Low-density parity-check codes*, IEEE Transactions on Information Theory, vol. 8, pp. 21-28, January 1962.

[33] R. G. Gallager, *Information Theory and Reliable Communications*, Wiley 1968.

[34] J. Garcia-Frias & J. D. Villasenor, *Joint Turbo Decoding and Estimation of Hidden Markov Sources*, IEEE Journal on Selected Areas in Communications, vol. 19, no. 9, pp. 1671-1679, September 2001.

[35] R. Garello & G. Montorsi & S. Benedetto & G. Cancellieri, *Interleaver Properties and Their Applications to the Trellis Complexity Analysis of Turbo Codes*, IEEE Transactions on Communications, vol. 49, no. 5, pp. 793-807, May 2001.

[36] N. Goertz, *On the Iterative Approximation of Optimal Joint Source-Channel Decoding*, IEEE Journal on Selected Areas in Communications, vol. 19, pp. 1662-1670, September 2001.

[37] R. M. Gray & Y. Lindo, *Vector quantizers and predictive quantizers for Gauss-Markov sources*, IEEE Transactions on Communications, vol. 30, pp. 381-385, February 1982.

[38] R. M. Gray & D. L. Neuho, *Quantization*, IEEE Transactions on Information Theory, vol. 44, No. 6, Oct. 1998.

[39] A. Guyader & E. Fabre & C. Guillemot & M. Robert, *Joint Source-Channel Turbo Decoding of Entropy-Coded Sources*, IEEE Journal on Sellected Areas in Communications, vol. 19, no. 9, pp. 1680-1696, September 2001.

[40] J. Hagenauer, *Rate-compatible punctured convolutional codes (RCPC codes) and their applications*, IEEE Transactions on Communications, vol. 36, no. 4, pp. 389-400, April 1988.

[41] J. Hagenauer, *Source-Controlled Channel Decoding*, IEEE Transactions on Communications, vol. 43, no. 9, pp. 2449-2457, September 1995.

[42] E. K. Hall & S. G. Wilson, *Stream-Oriented Parallel Concatenated Convolutional Codes*, Information Theory Workshop, 1998.

[43] E. K. Hall & S. G. Wilson, *Stream-Oriented Turbo Codes*, IEEE Transactions on Information Theory, vol. 47, pp. 1813-1831, July 2001.

[44] L. Hanzo & T. H. Liew & B. L. Yeap, *Turbo Coding, Turbo Equalization and Space-Time Coding*, New York Wiley/IEEE, 2002.

[45] C. Heegard & S. B. Wicker, *Turbo Coding*, Kluwer Academic Publishers, Massachussets, 1999.

[46] W. Henkel & L. Jusif & J. Sayir, *A Pipelined Turbo Decoder with Random Convolutional Interleaver*, 2003.

[47] T. Hindelang & S. Heinen & J. Hagenauer, *Source controlled channel decoding: Estimation of correlated parameters*, Proceedings of International ITG Conference on Source and Channel Coding, pp. 251-258. January 2000.

[48] T. Hindelang & T. Fingscheidt & N. Seshadri & R. V. Cox, *Combined source/channel (de-)coding: can a priori information be used twice?*, Proceedings of the IEEE International Conference on Communications (ICC 2000), vol. 3, pp. 1208-1212, June 2000.

[49] T.Hindelang, *Source-controlled channel encoding and decoding for mobile communications*, Ph.D. thesis, Munich University of Technology, 2001.

[50] J. Hokfelt & O. Edfors & T. Maseng, *On the Theory and Performance of Trellis Termination Methods for Turbo Codes*, IEEE Journal on Selected Areas in Communications, vol. 19, no. 5, pp. 838-847, May 2001.

[51] R. Johannesson & K. Sh. Zigangirov, *Fundamentals of Convolutional Coding*, IEEE Press, New York, 1998.

[52] J. Kim & J. Pearl, *A computational model for combined causal and diagnostic reasoning in inference systems*, Proceedings of the Eighth International Joint Conference on Artificial Intelligence, pp. 190-193, 1983.

[53] F. R. Kschischang & B. J. Frey, *Iterative decoding of compound codes by probability propagation in graphical models*, IEEE Journal on Selected Areas in Communication, vol. 16, pp. 219-230, 1998.

[54] F. R. Kschischang & B. J. Frey & H.-A. Loeliger, *Factor graphs and the sum-product algorithm*, IEEE Transactions on Information Theory, vol. 47, no. 2, pp. 498-519, February 2001.

[55] H. Kumazawa & M. Kasahara & T. Namekawa, *A construction of vector quantizers for noisy channels*, Electronics and Engineering in Japan, vol. 67-B, pp. 39-47, January 1984.

[56] A. J. Kurtenbach & P. A. Wintz, *Quantizing for noisy channels*, IEEE Transactions on Communications, vol. 17, pp. 291-302, April 1969.

[57] S. Lin & D. J. Costello Jr., *Error Control Coding: Fundamentals and Applications*, Prentice-Hall, New Jersey, 1983.

[58] Y. Linde & A. Buzo & R. Gray, *An Algorithm for Vector Quantizer Design*, IEEE Transactions on Communications, vol. 28, January 1980.

[59] D. J. C. MacKay & R. M. Neal, *Near Shannon limit performance of low-density parity-check codes*, Electronics Letters, vol. 32, pp. 1645, August 1996.

[60] D. J. C. MacKay, *Good error-correcting codes based on very sparse matrices*, IEEE Transactions on Information Theory, vol. 45, pp. 399-431, March 1999.

[61] F. J. MacWilliams & N. J. A. Sloane, *The Theory of Error-Correcting Codes*, Amsterdam, North Holland, 1977.

[62] B. Masnick & J. Wolf, *On linear unequal error protection codes*, IEEE Transactions on Information Theory, vol. 13, pp. 600-607, October 1969.

[63] J. L. Massey, *Joint source and channel coding*, Communication Systems and Random Process Theory, pp. 279-293, 1978.

[64] R. J. McEliece & D. J. C. Mackay & J.-F. Cheng, *Turbo decoding as an instance of Pearl's belief propagation algorithms*, IEEE Journal on Selected Areas in Communications, vol. 16, pp. 140-152, February 1998.

[65] D. J. Miller & M. Park, *A sequence-based approximate MMSE decoder for source coding over noisy channels using discrete hidden Markov models*, IEEE Journal Transactions on Communications, vol. 46, pp. 222-231, February 1998.

[66] G. Montorsi & F. Pollara, *On the design of turbo codes*, TDA Progress Report, vol. 42-123, November 1995.

[67] H. Niu & M. Shen & J. A. Ritcey & H. Liu, *Iterative Channel Estimation and LDPC Decoding over Flat-Fading Channels: A Factor Graph Approach*, Conference on Information Sciences and Systems, March 2003.

[68] J. Pearl, *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*, 1988.

[69] L. C. Perez & J. Seghers & D. J. Costello, *A distance spectrum interpretation of turbo codes*, IEEE Transactions on Information Theory, vol. 42, no. 6, pp. 1698-1709, November 1996.

[70] A. Q. Pham & L. L. Yang & L. Hanzo, *Joint optimization of iterative source and channel decoding using over-complete source-mapping*, Proceedings of the 66th IEEE Vehicular Technology Conference, pp. 1072-1076, September 2007.

[71] J. K. Proakis, *Digital Communications*, McGraw-Hill, Boston, 2000.

[72] J. L. Ramsey, *Realization of optimum interleavers*, IEEE Transactions on Information Theory, vol. 16, pp. 338-345, May 1970.

[73] T. J. Richardson & R. L. Urbanke, *The capacity of low-density parity-check codes under message-passing decoding*, IEEE Transactions on Information Theory, vol. 47, no. 2, pp. 599-618, February 2001.

[74] A. Ruscitto & E. M. Biglieri, *Joint Source and Channel Coding Using Turbo Codes Over Rings*, IEEE Transactions on Communications, vol. 46, no. 8, pp. 981-984, August 1998.

[75] C. E. Shannon, *A mathematical theory of communication*, The Bell System Technical Journal, vol. 27, pp. 379-423 & pp. 623-656, 1948.

[76] M. Shen & H. Niu & H. Liu, *Iterative Receiver Design in Rayleigh Fading Using Factor Graph*, Vehicular Technology Conference, 2003.

[77] M. Sipser & D. A. Spielman, *Expander Codes*, IEEE Transactions on Information Theory, vol. 42 pp. 1710-1722, 1996.

[78] M. Skoglund, *On channel-constrained vector quantization and index assignment for discrete memoryless channels*, IEEE Transactions on Information Theory, vol. 45, no. 7, pp. 2615-2622, July 1999.

[79] R. M. Tanner, *A recursive approach to low-complexity codes*, IEEE Transactions on Information Theory, vol. 27, pp. 533-547, September 1981.

[80] M. Tuechler & R. Koetter & A. C. Singer, *Graphical models for coded data transmission over inter-symbol interference channels*, 5th International ITG Conference on Source and Channel Coding (SCC), January 2004.

[81] A. J. Viterbi, *Error bounds for convolutional codes and an asymptotically optimum decoding algorithm*, IEEE Transactions on Information Theory, Volume 13 pp. 260-269, April 1967.

[82] A. J. Viterbi, *Convolutional codes and their performance in communication systems*, IEEE Transactions on Communications, vol. 19, pp. 751-772, October 1971.

[83] A. J. Viterbi & J. K. Omura, *Principles of Digital Communication and Coding*, McGraw-Hill, New York, 1979.

[84] N. Wiberg & H. A. Loeliger & R. Kotter, *Codes and iterative decoding on general graphs*, European Transactions on Telecommunications, vol. 6, no. 3, pp. 513-525, 1995.

[85] N. Wiberg, *Codes and decoding on general graphs*, Ph.D. thesis, Linköping University, 1996.

[86] S. B. Wicker, *Error Control Systems for Digital Communication and Storage*, Prentice-Hall, New Jersey, 1995.

[87] A. P. Worthen & W. E. Stark, *On iterative receivers for non-coherent channels*, Proceedings of the IEEE International Symposium on Information Theory and Its Applications (ISITA'00), November 2000.

[88] A. P. Worthen & W. E. Stark, *Unified design of iterative receivers using factor graph*, IEEE Transactions on Information Theory, vol. 47, no. 2, pp. 849-853, February 2001.

[89] K. Zeger & A. Gersho, *Pseudo-Gray coding*, IEEE Transactions on Communications, vol. 38, No. 12. pp. 2147-2158, 1990.